DOCUMENT RESUME

ED 230 199 IR 010 722

AUTHOR Mayer, Richard E.
TITLE Diagnosis and Remediation of Computer Programming Skill for Creative Problem Solving. Volume 1: Description of Research Methods and Results. Final Report.
INSTITUTION California Univ., Santa Barbara.
SPONS AGENCY National Inst. of Education (ED), Washington, DC. Teaching and Learning Program.
PUB DATE Oct 82
GRANT NIE-G-80-0118
NOTE 234p.; For related document, see IR 010 723.
PUB TYPE Reports - Research/Technical (143)

EDRS PRICE MF01/PC10 Plus Postage.
DESCRIPTORS Calculators; *Computer Literacy; Computers; *Learning Processes; *Models; Problem Solving; *Programing; *Programing Languages; Teaching Methods
IDENTIFIERS *BASIC Programing Language; Learning Strategies; Users

ABSTRACT

This 5-section report summarizes the results of a project concerned with how novices learn to become creative educational computer users. Based on a cognitive analysis of elementary programming statements in BASIC and calculator language into conceptual units, the project builds on previous research on learning BASIC programming. A general description of the project is followed by a review of the literature on the psychology of computer programming, especially research in cognitive psychology related to beginning programming. The analysis of both a calculator language and BASIC into conceptual units and the evaluation of users' knowledge is then described. Data on the frequency of misconceptions in users' knowledge of calculator and BASIC languages are presented. Focusing on instructional techniques useful in the remediation of specific misconceptions of users' knowledge of calculator and BASIC languages, the final chapters explore the effectiveness of teaching mental models and whether use of the models enhances problem-solving performance. References are included in each section, and an appendix lists additional papers and publications which are products of the project under review (either published or in preparation). (LMM)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
This document has been reproduced as received from the person or organization originating it.
Minor changes have been made to improve reproduction quality.
Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

ED230199

Final Report

Grant NIE-G-80-0118

DIAGNOSIS AND REMEDIATION OF COMPUTER PROGRAMMING SKILL FOR CREATIVE PROBLEM SOLVING

Volume 1

Description of Research Methods and Results

October, 1982

Richard E. Mayer, Principal Investigator

The Program on Teaching and Learning, U. S. National Institute of Education, funded this project through Grant NIE-G-80-0118 to the University of California, Santa Barbara.

IR010122

Outline of Volume 1

Chapter 1. Introduction 1

Chapter 2. Literature Review. 8

Chapter 3. Task Analysis, Development of Evaluation Instruments, and Diagnosis of Bugs for Calculator Language. 68

Chapter 4. Task Analysis, Development of Evaluation Instruments, and Diagnosis of Bugs for the BASIC Language 127

Chapter 5 Effect of Instruction on Remediation of Bugs and Problem Solving for Calculator Language. 167

Chapter 6. Effects of Instruction on Remediation of Bugs and Problem Solving for the BASIC Language 194

Appendix 1. List of Publications and Papers. 208

CHAPTER 1

INTRODUCTION

Focus of Project

This project is concerned with how novices learn to become creative users of electronic computers. In particular, this project contributes to what Scheiderman (1980) calls "software psychology" -- a theory of how humans think about and use computers. This project is based on a cognitive analysis of elementary programming statements in BASIC and Calculator Language into conceptual units, similar to techniques for analyzing arithmetic computational skill (Brown & Burton, 1978; Groen & Parkman, 1972; Resnick, 1976). In addition, this project builds on previous research on learning BASIC computer programming (Mayer, 1979).

The main goals of the project are the following:

(1) Analysis of the "bugs" in people's knowledge of elementary computer programming. This project specifies a user's knowledge in terms of specific "bugs" in his or her understanding of each statement or command. By using a more detailed level of analysis than has been traditionally used, we have been able to make specific diagnoses of what users do not know. This technique is similar in some respects to that used by Brown and Burton (1978) in analyzing procedural "bugs" in arithmetic.

(2) Remediation of common "bugs" in computer knowledge through instruction. Once a user's knowledge is described in terms of "bugs" in his or her understanding of statements, it is possible to develop remedial training that focuses on the missing concepts. As part of remedial instruction, we have used concrete models of the computer which allow us to show the flow of information for each line of code in a concrete and visual way. Our instruction helps the users to see what goes on "inside the computer" for each statement.

(3) Determination of techniques for helping students to become creative

users of computers. A third goal of this project is to determine whether the diagnosis and remediation techniques outlined above will enhance the problem-solving performance of our students. This project provides both a practical and a theoretical test of the usefulness of cognitive analysis techniques in a real learning and problem-solving task. In particular, this project has demonstrated that instruction in specific "mental models" has a positive influence on problem-solving performance in programming.

Rationale of Project

Some knowledge of computer programming is rapidly becoming a survival skill. Elementary computer programming is being incorporated into science and mathematics curricula of many schools, and as part of the technical training for many adults. Recently, Carnegie-Mellon University has announced that all students will be required to buy computer terminals as part of their college educational materials. With increasing improvements in computer power, coupled with decreasing costs, there are indications that computers will become a common feature of our nation's classrooms.

In spite of tremendous achievements in hardware development and the advances in producing educational software, there has been comparatively little basic research on how to teach computer programming. Other topics in mathematics and science have enjoyed long histories of instructional research, but computer programming is so new that it has not enjoyed a similar research emphasis.

Since computers are becoming a part of many classrooms in our nation, and since elementary programming is becoming a part of math and science curriculum, there is a need to better understand how beginners learn to use computers. This project provides new information concerning the processes by which novices learn computer programming, and how to help novices become

productive users of computers.

Organization of the Final Report

This Final Report describes the methods and results of the project. There are five major sections of the report, covering each of the tasks listed in the project proposal.

Chapter 2 presents a review of literature concerning the psychology of computer programming. The review focuses on research in cognitive psychology that is relevant for the study of how novices learn programming languages.

Chapter 3 describes how a calculator language can be analyzed into conceptual units and how users' knowledge can be evaluated. In addition, Chapter 3 presents data concerning the frequency of major "bugs" or misconceptions in users' knowledge of calculators.

Chapter 4 describes how BASIC can be analyzed into conceptual units and how users' knowledge can be evaluated. In addition, Chapter 4 presents data concerning the frequency of major "bugs" (or misconceptions) in users' knowledge of BASIC.

Chapter 5 focuses on instructional techniques that may be useful in the remediation of specific "bugs" in users' knowledge of calculator language. In particular, this chapter explores whether students can be taught to use "mental models" and whether use of the models enhances problem-solving performance.

Chapter 6 focuses on instructional techniques that may be useful in the remediation of user's "bugs" concerning BASIC statements. In particular, this chapter explores whether students can be taught to use various "mental models" for BASIC, and whether use of the models enhances problem-solving performance.

These six chapters constitute the first volume of this final report. The second volume summarizes the same research, but does so in a way more

appropriate for practitioners.

Project Accomplishments

The Major accomplishments of this project have been the following:

Task analysis. We have developed and refined an analysis of BASIC and Calculator Language. The analysis yields a set of elementary actions that go on inside the computer, and which are important for the user's understanding of the language.

Development of Evaluation Instruments. We developed structured questionnaires to assess a user's knowledge of the statements of BASIC and Calculator Language. Through several studies, we refined the evaluation instruments so that we are now able to measure user's conceptions of computer languages.

Diagnosis of bugs. Using the evaluation instruments, we have identified typical bugs in users' understanding of BASIC and Calculator Language. For example, a high proportion of users think that LET X = A + B means that an equation is stored in memory. As another example, a high proportion of users think that READ A means that the computer prints out the value stored in memory space A. A major product of this diagnosis effort is a list of standard misconceptions for each statement, based on student performance.

Remediation of bugs. We have developed and implemented training procedures that are aimed at correcting specific bugs in a user's knowledge. For both BASIC and Calculator Language, we have developed instructional techniques based on our task analysis. These techniques have strong and reliable effects on elimination of bugs in user's understanding of statements.

Effects on problem solving. Finally, we have studied the effects of user's knowledge of BASIC on actual problem solving in BASIC. The results indicate that students' problem-solving performance is enhanced when bugs in

understanding of statements are minimal. Thus, we have obtained evidence that user's knowledge of BASIC statements at a "transaction level" is related to superior problem-solving performance.

Project Implications

This project provides new information concerning how students learn computer programming statements. In particular, this project has documented the existence of users' misconceptions of computer programming statements, and has examined instructional techniques for remediating these misconceptions. The results have implications both for cognitive theory and for the teaching of computer programming in American schools.

The results indicate that "hands-on experience" does not necessarily lead to understanding of programming languages. Many students developed serious bugs in spite of hands-on experience. Some direct instruction is required concerning the underlying conceptual events and ideas in computer programming.

The results indicate that it is possible to provide direct instruction concerning "what goes on inside the computer" for each statement. This level of instruction is needed when the goal of instruction is creative problem solving.

The results indicate that remediation efforts could be directed at specific bugs rather than overall mastery.

"What is learned" in computer programming involves both specific facts and a "mental model" of the system. Students need help in developing productive mental models, including direct instruction about models.

References

Brown, J. S. and Burton, R. R. Diagnostic models for procedural bugs in basic mathematical skills. Cognitive Science, 1978, 2, 155-192.

Groen, G. J. and Parkman, J. M. A chronometric analysis of simple addition. Psychological Review, 1972, 79, 329-343.

Mayer, R. E. A psychology of learning BASIC. Communications of the ACM, 1979, 22, 589-593.

Resnick, L. B. Task analysis in instructional design: Some cases from mathematics. In D. Klahr (Ed.), Cognition and instruction. Hillsdale, N. J.: Erlbaum, 1976.

Shneiderman, B. Software Psychology: Human Factors in Computer and Information Systems. New York: Winthrop, 1980.

CHAPTER 2

LITERATURE REVIEW

Note

This review has been published as the following citation:

Mayer, R. E. The psychology of how novices learn computer programming. Computing Surveys, 1981, 13, 121-141.

For purposes of wider dissemination, the above article was also reprinted in the following edited book:

Curtis, B. Human Factors in Software Design. Los Angeles: IEEE Computer Society Press, 1981.

In addition, the above article was reprinted in the Japanese computer magazine, Byte.

This article was also published as a technical report:

Mayer, R. E. Psychology of computer programming for novices. Technical Report No. 81-2. Santa Barbara: University of California, 1981.

INTRODUCTION

This article focuses on the question, "What have we learned about how to increase the novice's understanding of computers and computer programming?" In particular, this paper reviews ideas from cognitive and educational psychology that are related to the problem of how to teach non-programmers to use computers. Since people who are not professional programmers will have to learn how to interact with computers, an important issue concerns how to foster meaningful learning of computer concepts by novices.

Meaningful learning is viewed as a process in which the learner connects new material with knowledge that already exists in memory (BRAN79). The existing knowledge in memory has been called "schema" and the process of connecting new information to it has been called "assimilation." However, there is not yet agreement concerning the specific mechanisms that are involved in "assimilation to schema" (ANDE77, AUSU77, BARD32, KINT74, MINS75, RUME75, SCHA77, THOR77).

Figure 1 provides a general framework for discussion the process of meaningful learning (or assimilation to schema) of technical information (MAYE75a, MAYE79a). In the figure, the human cognitive system is broken down into:

short term memory -- a temporary and limited capacity store for holding and manipulating information, and

long term memory -- a permanent, organized, and unlimited store of existing knowledge.

New technical information enters the human cognitive system from the outside and must go through the following steps for meaningful learning to occur:

(a) Reception. First the learner must pay attention to the incomi information so that it reaches short term memory (as indicated by arrow a).

(b) Availability. Second, the learner must possess appropriate prerequisite concepts in long term memory to use in assimilating the new information (as indicated by point b.)

(c) Activation. Finally, the learner must actively use this prerequisite knowledge during learning so that the new material may be connected with it (as indicated by arrow c from long term memory to short term memory).

Thus, in the course of meaningful learning, the learner must come into contact with the new material (by bringing it into working memory), then must search long term memory for what Ausubel (AUSU68) calls "appropriate anchoring ideas" or "ideational scaffolding," and then must transfer those ideas to working memory so they can be combined with new incoming information. If any of these conditions is not met, meaningful learning cannot occur; and the learner will be forced to rotely memorize each piece of new information as a separate item to be added to memory. The techniques reviewed in this article are aimed at insuring that the availability and activation conditions are likely to be met.

Insert Figure 1 about here

The goal of this article is to explore techniques for increasing the novice's understanding of computer programming by exploring techniques that activate the "appropriate anchoring ideas." Two techniques reviewed in this paper are: (1) providing a familiar concrete model of the computer, and

(2) encouraging learners to put technical information into their own words. Each technique is an attempt to foster the process by which familiar existing knowledge is connected with new incoming technical information. For each technique, a brief rationale is presented, examples of research are presented, and an evaluative summary is offered.

1. UNDERSTANDING OF TECHNICAL INFORMATION BY NOVICES

1.1 Definitions

For present purposes, understanding is defined as the ability to use learned information in problem solving tasks that are different from what was explicitly taught. Thus, understanding is manifested in the user's ability to transfer learning to new situations. Novices are defined as users who have had little or no previous experience with computers and who do not intend to become professional programmers, and thus lack specific knowledge of computer programming.

1.2 Distinction Between Understanding and Rote Learning

The Gestalt psychologists (WERT59, KATO42, KOHL25) distinguished between two ways of learning how to solve problems--"rote learning" vs. "understanding." With respect to mathematics learning, for example, there is often a distinction made between "getting the right answer" and "understanding what you are doing."

In a classic example, Wertheimer suggests that there are two basic ways to teach a child how to find the area of a parallelogram (WERT59). One method involves dropping a perpendicular line, measuring the height of the perpendicular, measuring the length of the base, and calculating area by use of the formula, Area = Height x Base. Wertheimer calls this the "rote learning" or "senseless" method, because the student simply memorizes a formula and a procedure. The other method calls for the student to visually explore the parallelogram until the student sees that you could cut a triangle from one end,

put it on the other end, and form a rectangle. Since the student already knows how to find the area of a rectangle, the problem is solved. Wertheimer calls this method "structural understanding" or "meaningful apprehension of relations," since the learner has gained insight into the structure of parallelograms.

According to Wertheimer, if you give a test involving parallelograms like the one used during instruction, both groups of children will perform well. However, if you give a transfer test that involves unusual parallelograms then the rote learners will say, "We haven't had this yet," while the understanders will be able to derive answers. Thus, the payoff for understanding comes not in direct application of the newly learned material, but rather in transfer to new situations. This example suggests that when creative use of new technical information is the goal, it is important to use methods that foster understanding.

2.0 DO CONCRETE MODELS AID MEANINGFUL LEARNING OF COMPUTER PROGRAMMING?

2.1 Statement of the Problem

Since novices lack domain specific knowledge, one technique for improving their understanding of new technical information is to provide them with a framework that can be used for incorporating new information. This technique is aimed at insuring availability of knowledge in long term memory (See Figure 1). The present section explores the effects of concrete models on people's understanding and learning of new technical information such as computer programming. The major research questions concern how concrete models influence the learning process and how to choose an effective model.

2.2 Concrete Models in Mathematics Learning

One technique for providing the appropriate prerequisite knowledge is the use of familiar, concrete models. For example, Brownell & Moser (BROW49) taught third graders how to use a subtraction algorithm, using two different

methods. One group of several hundred children was taught by using concrete objects like bundles of sticks. For these children, concepts like "borrowing" and "place value" were described in terms of rearranging bundles of sticks into groups of tens. The other group was taught in a "purely mechanical rote fashion"; these children were explicitly given the rules for subtraction at the start and given plenty of "hands on" experience in executing the procedures on standard two digit subtraction problems. Although both groups of students learned to perform equally well on standard two digit subtraction problems, the students who learned with bundles of sticks performed better on tests involving transfer problems (e.g., more complicated subtraction problems).

In current instructional practice, manipulatives, such as coins or sticks or blocks, are used in mathematics teaching in order to make computational procedures more concrete (WEAV72, RESN80). In a careful set of interviews with children who were learning to subtract, Resnick & Ford (RESN80) noted that children often invented a concrete model to help them understand the procedure. Since computer programming shares many of the characteristics of computational procedures in mathematics, it seems possible that the use of manipulatives in computer programming might be as successful as in mathematics.

2.3 Models, Titles, and Advance Organizers in Text

There is also encouraging evidence that similar techniques may be used to increase the meaningfulness of technical information presented in text. For example, Bransford & Johnson (BRAN72) presented the following passage to subjects:

> The procedure is actually quite simple. First you arrange items into different groups. Of course, one pile may be sufficient depending on how much there is to do. If you

have to go somewhere else due to lack of facilities that is the next step; otherwise, you are pretty well set. It is important not to overdo things. In the short run this may not seem important, but complications can easily arise. A mistake can be expensive as well. At first, the whole procedure will seem complicated. Soon, however, it will become just another facet of life. It is difficult to foresee any end to the necessity for this task in the immediate future, but then, one never can tell. After the procedure is completed one arranges the materials into different groups again. Then they can be put into their appropriate places. Eventually they will be used once more and the whole cycle will have to be repeated. However, this is part of life.

Subjects who read this passage, without a title, rated it low in comprehensibility (2.3 on a 7 point scale) and recalled an average of only 2.8 out of 18 ideas from the passage. However, some subjects were given a description of the topic--washing clothes--before the passage. These subjects rated the passage much higher in comprehensibility (4.5 on a 7 point scale) and recalled more than twice as much information (5.8 idea units out of 18). In addition, a third group was given the washing clothes topic after the passage was presented. However, this group performed at about the same low level as the subjects who were given no topic, (rating the passage at 2.1 in comprehension and recalled an average of 2.7 idea units). Similar studies (BRAN72, DOOL71, DOOL72) also found that students' recall of ambiguous and technical passage was enhanced when an organizing title or diagram or sentence was given prior to reading. However,

these techniques did not have the same facilitating effect when presented after the student had read the passage. These results suggest that the learner must have an appropriate assimilative set available at the time of learning. Even though the same total amount of information may be presented, the students' ability to recall and use the information in the passage is much higher when the clarifying title or picture is given before rather than after reading.

Ausubel (AUSU68) has argued that learning of new technical prose may be enhanced by providing an advance organizer--a short expository introduction, presented prior to the text, containing no specific content from the text, but providing the general concepts and ideas that can be used to subsume the information in the text. The first advance organizer studies conducted by Ausubel and his colleagues in the early 1960's (AUSU60, AUSU63, AUSU68), provided some support for this assertion. For example, in a typical study (AUSU60), 120 college students read a 2500-word text on metallurgy after reading either a 500-word advance organizer that presented the underlying framework for the information or a control 500-word historical passage. The advance organizer presented the abstract principles involved in the text. On a reading comprehension post-test covering the basic information in the passage, the advance organizer group performed significantly better than the control group, with scores of 47% correct versus 40% correct, respectively.

More recently, reviews of the advance organizer literature reveal that advance organizers tend to have their strongest effects in situations where learners are unlikely to already possess useful prerequisite concepts--namely, for technical or unfamiliar material, for "low ability" or inexperienced students, and when the test involves transfer to new situations (MAYE79a, MAYE79b). For example, to study the effects of advance organizers on different kinds of materials, Lesh (LESH76) asked 48 college students to watch a four-

hour videotape on finite geometry. An organizer that gave concrete examples and models was provided either before or after instruction. The instructional lesson was organized either in a order of increasing difficulty (hierarchical order) or in an order that repeated key concepts and related new material to previous material (spiral order). Results of a standard post-test indicated that the advance organizer group outperformed the post-organizer group for the hierarchical unit, but the difference was much less for the spiral unit. Similar treatment x material type interactions were obtained using social studies lessons (SCHU75) and mathematics lessons (GROT68). Similarly, Raye (RAYE73) reported that the title biasing effects obtained by Bransford & Johnson (BRAN72) with the washing clothes passage were eliminated when the passage was made more concrete and familiar. Thus, there is consistent evidence that organizers have stronger effects for unfamiliar, abstract information than for familiar, concrete information.

In a study investigating the effects of advance organizers on students with high and low ability (or knowledge), physics material was taught to high school students (WEST76). Advance organizers, consisting of concrete models, tended to improve test performance of low ability students but had a much smaller effect for high ability subjects. Similar group x ability interactions were obtained by several other researchers (RING71, FITZ63, AUSU62, AUSU61, AUSU63, AUSU77, SMIT69). Thus, there is evidence that advance organizers have a stronger effect on low knowledge or low ability learners as compared to high knowledge or high ability learners.

Finally, in studies involving transfer tests (i.e. problems that are different from those in instruction), there is consistent evidence that advance organizers have a stronger effect on transfer performance than on simple retention. For example, this pattern was obtained with material on mathematical

topology (SCAN67), number bases (GROT68, MAYE77), and an imaginary science (MERR66).

Many of the apparent conflicts in the advance organizer literature (BARN75, LAWT77) can be accounted for by the idea that advance organizers find a way of connecting new information with existing knowledge--organizers are not needed for familiar material, experienced learners, or when the test does not involve jransfer.

While there is at present no foolproof procedure for generating useful advance organizers, a careful review of the existing literature suggests the following guidelines (MAYE79a): (1) The organizer should allow the reader to generate all or some of the logical relations in the text. (2) The organizer should provide a means of relating the information in the text with existing knowledge. (3) The organizer should be familiar to the learners. (4) The organizer encourages the learner to prerequisite knowledge that the learner would not normally have used. To date, advance organizers have been most effectively used in mathematics and science topics (MAYE79a).

Royer and his colleagues (ROYE75, ROYE76) have demonstrated that concrete models may serve as effective advance organizers in learning of new scientific information. In their studies, subjects read two passages, such as a passage on electrical conductivity followed by a passage on heat flow. For some subjects, the first passage contained several concrete analogies, such as electrical conduction being described as a chain of falling dominoes. For other subjects, the first passage presented the same information in abstract form without any concrete analogies. Reading of the second passage was facilitated if students had been given concrete models in the first passage (e.g., recall of the information in the second passage was about twice that of control groups). Apparently, the models presented in the first passage could be used

by learners during the reading of the second passage to help relate the technical terms to familiar concepts.

Similarly, White & Mayer (WHIT80) analyzed physics textbooks to determine how concrete models were used. For example, many textbooks explain Ohm's Law by describing water flowing in pipes, or a boy pushing a heavy load up an inclined street, or electron flow through a circuit. Recent results (MAYE80) show that when concrete analogies are embedded in a technical text, novices tend to perform best on recalling these familiar models and tend to recognize the information adjacent to the model in the text.

2.4 Concrete Models in Computer Programming

In previous sections, research was presented concerning the role of manipulatives in mathematics instruction, titles and pictures in remembering ambiguous passages, and advance organizers and models in science text. In each case there was evidence that these techniques serve to provide the learner with appropriate anchoring knowledge that is required for comprehension of new technical information. The present section focuses on research related specifically to computer programming.

DuBoulay and his colleagues (DUBO76, DUBO80) have provided a concrete model for teaching LOGO to children. The model consists of a conceptual LOGO machine with concrete memory locations, switches and work space, which allow the learner to "work" the machine.

DuBoulay and his colleagues have argued that there are two basic approaches to learning to interact with a computer. The first approach could be called the black box approach. In this approach the user developes the attitude that the computer is a black box--you put in commands and data and out comes the answer as if by magic. The mechanisms by which the computer operates are hidden from the user, and the user is likely to assume that computers are just

not understandable. Such users are likely to memorize algorithms that "work"--i.e., that generate the desired answers. However, such users are not able to relate the commands to an understanding of what goes on inside the black box.

The second approach is what can be called the glass box approach. In this approach, the user attempts to understand what is going on inside the computer. Each command results in some change in the computer and these changes can be described and understood. The level of description need not--indeed should not--be at the "blood and guts" level. Users do not need to become electronics experts. There is an appropriate level of description that Mayer (MAYE79c) refers to as the "transaction level." Similarly, DuBoulay et al. (DUBO80) offer two important properties for making hidden operations of a language more clear to a novice: (1) simplicity--there should be a "small number of parts that interact in ways that can be easily understood," and (2) visibility--novices should be able to view "selected parts and processes" of the model "in action." The LOGO model appears to fit these specifications because it is a simple, familiar model of the computer operations involved in LOGO; in short, it allows the user to develop intuitions about what goes on inside the computer for each line of code. Unfortunately, however, DuBoulay and his colleagues have not provided empirical tests concerning whether the LOGO machine model actually influences the problem solving performance of new learners, as compared to "traditional" methods that emphasize only "hands on experiences."

2.5 Effects of Models on Transfer Performance

In order to provide some information concerning the effects of concrete models on learning computer programming, a series of studies was conducted (MAYE75b). In the studies, subjects were either given a concrete model of the computer or not. Then subjects read a manual on a BASIC-like language and took a transfer test on the material.

Method. Figure 2 shows the model of the computer that was used to explain elementary BASIC-like statements to novices. The model provides concrete

---

Insert Figure 2 about here

---

analogies for four major functional units of the computer: (1) input is represented as a ticket window in which data is lined up waiting to be processed and is placed in the finished pile after being processed, (2) output is represented as a message note pad with one message written per line, (3) memory is represented as an eraseable scoreboard in which there is natural destructive read-in and non-destructive read-out, and (4) executive control is represented as a recipe or shopping list with a pointer arrow to indicate the line being executed. This model is similar to DuBoulay's model of the LOGO machine in the way it makes the basic operations of the computer visible to the learner. A 2 x 3 foot diagram containing these parts, and a brief one page description were provided to subjects in the "model group" (see Figure 2) but no model was given to the "control group."

All subjects then were given a 10-page manual that described seven statements modified from BASIC and FORTRAN. (see Table 1). For each statement, the manual presented the statement, provided the grammar rules for the statement (e.g., definitions of legal address names), and gave an example of the statement as it might occur in a line of a program. Subjects in both groups were given the same manual to read at their own rates, averaging 20 to 30 minutes.

Following reading, the same test was given to all subjects. The test consisted of six types of problems: (1) generate-statement problems gave a problem in English and required a one statement program as the solution,

(2) generate-nonloop problems gave a problem in English and required a short non-looping program for solution, (3) generate-looping problems gave a problem in English and required a looping program for solution, (4) interpret-statement problems gave a single statement program and asked the student to describe what the computer would do, (5) interpret-nonloop gave a non-looping program and asked for a description of what the computer would do (6) interpret-looping problems gave a looping program and required a description of what the computer would do. Examples of the six problems are given in Table 2.

Results. The proportion correct response by type of problem for each of the treatment groups is given in Table 3. As can be seen, the control group performs as well or better on problems that are very much like the material in the instructional manual, e.g., generate-statement and generate-nonloop. However, on problems that require moderate amounts of transfer[1]--e.g., generate-loop and the shorter interpret problems--the model group excels. Both groups do poorly on the very complex interpret-looping programs. The difference in the pattern of performance is consistent with earlier results in other domains in which models enhance transfer performance but not simple retention of presented material. Apparently, the model provided an assimilative context in which novices could relate new technical information in the booklet to a familiar analogy. This learning process resulted in a learning outcome that supported some transfer.

Insert Tables 1, 2 and 3 about here

2.6 Locus of the Effect of Models

One problem with the above study is that the model subjects received more information than the controls. However, assimilation theory (see Introduction)

predicts that presenting a model prior to learning will enhance learning because it provides a meaningful context, but presenting the model after the text will not enhance learning because students will have already encoded the material in a rote way. In further studies (MAYE76a) subjects read the same BASIC-like manual, but some subjects were shown a concrete model of the computer before reading while others were shown the same model after reading the manual. Thus, subjects in the before group (i.e., those who received the model first) were able to use the model while encoding the material in the text, but the after group (i.e., receiving the model last) was not.

Method. The booklet, model and test were similar to those used in the previous experiment. The before group received the model, then the booklet, then the test. The after group received the booklet, then the model, and then the test.

Results. The proportion of correct answers by type of problem for the two groups is given in Table 4. As can be seen, the after group (like the controls in the previous study) excels on retention-like problems (i.e., generation-statement and generation-nonloop), but the before subjects excel on problems requiring creative transfer to new situations (i.e., generation-loop, interpretation-statement, interpretation-nonloop). Thus, these results provide further support for the claim that subjects who use a concrete model during learning develop learning outcomes that support broader transfer. As predicted, the locus of the effect is before rather than after instruction.

Insert Table 4 about here

2.7 Effects of Models on Recall Performance

The above studies used transfer tests as a measure of what is learned under different instructional techniques. Another technique involves asking subjects to try to write down all they can remember about certain statements. In a follow-up study (MAYE80) subjects read the same manual and were given the model either before or after reading as in the previous study. However, as a test, subjects were asked to recall all they could about portions of the manual.

Method. The same booklet and model were used as in the previous experiments, with some minor modifications. The before group received the model, then the manual, then the recall test; the after group received the manual, then the model, and then the recall test.

Results. In order to analyze the recall protocols, the information in the manual was broken down into "idea units." Each idea unit expressed one major idea or action. There were three kinds of idea units in the manual: (1) conceptual idea units related to the internal operation of the computer, (2) technical idea units gave examples of code, and (3) format idea units gave grammar rules. Table 5 gives examples of each type of idea unit.

Table 6 shows the average number of correctly recalled idea units from each category for the two groups. As can be seen, the before group recalls more conceptual information while the after group recalls more technical and format information. This pattern is consistent with the idea that good retention requires recall of specific code, but good transfer requires understanding of conceptual ideas. Also, Table 6 shows that the before group included more intrusions about the model and about other idea units from other sections of the booklet, thus suggesting they integrated the information better. For example, an intrusion is, "An address is a slot in the memory scoreboard." The after group, however, included more vague summaries and connectives which served as "filler." For example, a connective is "And that's

(with very few operations) to a compute-2 program (with many different statements integrated into one large program). Table 8 lists the five different kinds of programs used.

Results. Table 9 gives the proportion of correct answers by type of problem for the two treatment groups. As can be seen, the control group performs as well as the model group on very simple problems like those in the manual, but the model group excels on longer problems that require creatively integrating all of the statements in the booklet. Thus, as in the studies with Basic-like materials, a familiar model serves to enhance performance on creative transfer when it is presented prior to technical instruction.

Insert Figure 3 and Tables 7, 8 and 9 about here

2.9 Ability

The pattern of results described above tended to be strongest for low ability subjects (MAYE75b) where ability is defined in terms of Mathematics SAT score. For example, for low ability subjects the advance organizer increased transfer test performance (55% correct) as compared to the control group (45% correct), but for high ability learners the advance organizer group performed more poorly than the control group (55% versus 62% correct, respectively). Apparently, high ability learners already possessed their own useful "models" for thinking about how a computer works, but low ability students are more likely to lack useful prerequisite knowledge.

2.10 Text Organization

The pattern of results described above also tended to be strongest when material was poorly organized (MAYE78). For example, the Basic-like manual was presented either in its original order or in a random order. In the random

how READ statements work." Thus, as with the transfer test, subjects given the model before learning show evidence of more integrated and conceptual learning of technical information.

Insert Tables 5 and 6 about here

2.8 Effects of Models on Transfer and Recall Using a Different Language

Although the above results are consistent and were obtained in a long series of studies, their generality is limited by the fact that just one type of language was used. Thus, a follow-up study (MAYE80a) was conducted using a file management language based on SEQUEL (GOUL74; REIS 77). The goal of this study is to determine whether the results from previous studies generalize to a new domain.

Method. Subjects read a manual that presented the file management language. For one group of subjects, the model group, the manual began with discussion of a concrete model and related each statement to the model (see Figure 3), but no model was given to the control group. The manuals were informationally equivalent. Each page of the booklet presented one of the eight statements shown in Table 7, along with examples of how the statement fit into a program. Figure 3 presents the concrete model that was used: long-term memory is represented as a file cabinet; the sorting function is represented as an in-basket, out-basket, and save basket; temporary memory is represented as an erasable scoreboard; executive control is represented as a list and pointer arrow; output is represented as a message pad. The entire model was presented on a 2 x 3 foot diagram, in order to enhance the learner's ability to visualize the system.

After reading the manual, all subjects took the same 20 item test. Problems varied in complexity from generating or interpreting a sort-1 program

order, presentation order of paragraphs was randomized. For the randomized version of the manual, the advance organizer group performed better on a transfer test than a control group (41% correct versus 31% correct, respectively); but for the logical version of the manual advance organizer group performed but did not outperform the control group (36% versus 44% correct, respectively). Apparently, the model is more useful when material is poorly structured because it helps the reader to hold the information together.

2.11 Conclusion

These results provide clear and consistent evidence that a concrete model can have a strong effect on the encoding and use of new technical information by novices. These results provide empirical support to the claims that allowing novices to "see the works" allows them to encode information in a more coherent and useful way (DUBO76, DUBO78). When appropriate models are used, the learner seems to be able to assimilate each new statement to his or her image of the computer system. Thus, one straightforward implication is: If your goal is to produce learners who will not need to use the language creatively, then no model is needed. If your goal is to produce learners who will be able to come up with creative solutions to novel (for them) problems, then a concrete model early in learning is quite useful. More research is needed in order to determine the specific effects of concrete models on what is learned, and to determine the characteristics of a useful model.

3.0 DOES STUDENT ELABORATION ACTIVITY AID MEANINGFUL LEARNING?

3.1 Statement of the problem

The previous section provided evidence that concrete models may influence learning of computer programming because they provide a familiar context for assimilating the new material. The second major technique for increasing

the meaningfulness of technical information is elaboration--encouraging the learner to explain the information in his or her own words and to relate the material to other ideas or concepts. Elaboration techniques may influence meaningful learning because they encourage the activation of existing knowledge that is relevant for comprehending the newly presented material, i.e., elaboration may affect the activation process (see Figure 1).

3.2 Putting It in Your Own Words

There is some evidence that asking subjects to put ideas into their own words during learning can enhance the breadth of learning. For example, Gagne & Smith (GAGN62) asked subjects to give a verbal rationalization for each step as they learned to solve a three disc version of the Tower of Hanoi problem (EWER32). These subjects took longer to learn than those who did not verbalize; however, they were able to transfer what they had learned to different problems, such as a six disc version, much more efficiently (e.g., 3.8 minutes to solution) than the non-verbalizers (e.g., 10.0 minutes to solution).

More recently, Wittrock (WITT74), has proposed the idea that "learning is a generative process"--i.e., learning occurs when the learner actively generates associations between what is presented and what he already has in memory. As an example, Wittrock (WITT74) presented a study in which elementary school children read a passage and either generated a one-sentence summary for each paragraph or did not. Recall by the students who generated summary sentences was nearly double that of the control group. Apparently, when students are actively encouraged to put information in their own words, they are able to connect the new information to existing knowledge.

Elaboration techniques have long been used by experimental psychologists to enhance the learning of paired associates (such as HOUSE-CASA). For example, when students are asked to actively form mental images or a sentence involving

word pairs, paired associate recall is greatly enhances (BOWE72, PAIV69). More recently, elaboration techniques have been used in school curricula (DANS78, WEIN78). For example, in studying human physiology, students are asked "How do arteries differ from veins?". Several researchers have argued that students should be given explicit training in "learning strategies" for actively processing new material (ONEI78).

The following is a series of studies that explore the role of elaboration techniques in learning computer programming. The main theme of this research is to determine how "putting it in your own words" influences the learning of a new computer language.

3.3 Effects of Model Elaboration on Transfer Performance

The first set of studies (MAYE80a) address the question of whether elaboration activity influences students' ability to engage in problem solving. In these studies, subjects learned a new computer programming language and either were or were not encouraged to describe what they learned in their own words by relating it to a concrete familiar situation.

Method. Subjects read an instructional manual covering an information management language similar to that described in the previous section (see Tables 7 and 8). For subjects in the model elaboration group, there was an elaboration page after each page in the manual while for subjects in the control group there was no elaboration exercise. The elaboration exercises asked the subject to describe the newly learned statement in terms of operations within a concrete model of the computer. Table 10 provides a typical exercise. Then, all subjects took the same 20 item problem solving test as described in the previous section.

Insert Table 10 and 11 about here

Results. Table 11 shows the proportion correct response by type of problem for the two groups.[2] As can be seen, the control group performs well on simple retention-like problems, but the model elaboration group performs considerably better on problems requiring creative transfer. Thus, there is evidence that requiring the learners to put technical information in their own words though relating the material to a familiar situation, results in broader learning outcomes. The results are similar to those given in Table 9, and suggest that model advance organizers and model elaboration have similar effects.

3.4 Effects of Comparative Elaboration on Transfer Performance

In the previous study, a concrete situation is presented and the learner is asked to relate the new information to it. However, the results are ambiguous in the sense that they may be attributed either to elaboration activity per se or to the fact that additional information (about the concrete model) was presented to the model elaboration group. The purpose of the present studies were to use a kind of elaboration activity that does not add new information (MAYE80a). Thus, a set of studies were conducted in which some subjects were asked to compare newly learned statements in their own words.

Method. Subjects read the same manual about an information management language as in the previous study. However, some subjects were given an elaboration page after each page in the booklet (comparative elaboration group), while for other subjects there was no elaboration (control group). The elaboration activity asked subjects to tell how two statements were similar and different, in their own words. Table 12 provides a typical exercise. Then, all subjects took the same test as in the previous study.

Results. Table 13 shows the proportion of correct answers by type of problem for the two groups. As can be seen, the control group excels on

retention-like problems but the comparative elaboration groups excels on the more complex transfer problems. Thus, there is evidence corresponding to that found in the model elaboration studies, that asking learners to put technical information in their own words (through making comparisons) results in broader learning which supports transfer.

Insert Tables 12 and 13 about here

3.5 Effects of Model and Comparative Elaboration on Recall

The previous studies suggest that elaboration activity can influence transfer performance. As a further test (MAYE80a) subjects were given manuals with either no elaboration questions, model elaboration questions, or comparative elaboration questions. It can be predicted that the elaboration subjects should recall more information that supports transfer--such as conceptual information--while the control group should recall more information about specific statements--such as technical information.

Method. As in the previous study, subjects read a manual explaining the information management language that contained either no questions (control group), model questions (model elaboration group), or comparative questions (comparative elaboration group). Then, subjects were asked to recall portions of the text.

Results. For purposes of scoring the recall protocols, the text was divided into idea units. Some of the idea units presented information about how the computer operated (conceptual idea units) and others emphasized the grammar and technical aspects of each statement (technical idea units). Table 14 shows the average number of idea units recalled by type for the three groups. As can be seen the control group recalls equal amounts of both types of information,

but the elaboration groups each tend to emphasize recall of conceptual as compared to technical information. These results are consistent with the results of the transfer studies, in that conceptual information is likely to be needed to support transfer.

Insert Table 14 about here

3.6 Effects of Note-taking on Transfer and Recall Performance

The foregoing series of studies provides some evidence that elaboration techniques influence the breadth of learning. However, the generality of the result is limited by the fact that just one type of manual was used and just two types of elaboration activity. In addition, previous studies did not control for amount of reading time. Thus, an additional series of studies (PEPE78) was conducted using a different language (a BASIC-like language) and a different elaboration activity (note-taking).

Method. Subjects watched a 20 minute videotape lecture describing seven BASIC-like statements similar to the manual described earlier. Some subjects were asked to take notes, by putting the basic information in their own words. Other subjects simply viewed the lecture without taking notes. As a test, subjects were given problems to solve or asked to recall portions of the lesson. Videotape presentations controlled for presentation time in the two groups.

Results. Table 15 gives the proportion of correct answers on generative problems (similar to those in the lesson) and on interpretation problems (which were not in the lesson). As can be seen, for low ability subjects (based on Mathematics SAT scores), there is a pattern in which note-taking helps performance on transfer but hurts performance on the retention-like problems. For high ability subjects, note-taking has no effect, presumably because high ability

learners already possess strategies for actively assimilating the new information.

Table 16 shows recall of the lecture by type of idea unit for the two groups. As can be seen, the note takers recall more conceptual information, but there is no difference between the groups in recall of technical information. Thus, the results are consistent with the model elaboration and comparative elaboration studies concerning the effects of asking subjects to put new technical information in their words during learning.

Insert Tables 15 and 16 about here

3.7 Conclusion

The goal of elaboration is to help the learner to be able to describe the key concepts in his own words, using his existing knowledge. Unfortunately, there is no fool-proof way to design useful elaboration activities. Emphasis on format or grammatical details and emphasis on errorless verbatim recall of statements will not produce the desired effects. The learner should be able to describe the effects of each program statement in his own words.

4.0 UNDERSTANDING COMPUTER PROGRAMMING

The previous sections have focused on the issue of how to teach novices. This section briefly examines the issue of what to teach. Greeno (GREE76) has argued that instruction for problem solving tasks should be based on cognitive objectives--statements of what the learner should have in his or her head at the end of instruction. Two major objectives that are relevant to enhancing a novice's understanding of computer programming are: knowledge for understanding a statement and knowledge for understanding a program.

4.1 Understanding a Statement

What does it mean to say that someone "understands" a certain statement? In a recent analysis of BASIC, each statement is described as a "transaction" (MAYE79c). A "transaction" consists of an action, an object, and a location in the computer. For example, the statement LET X = 5, consists of the following six transactions:

1. Find the number indicated on the right of the equals. (ACTION: Find; OBJECT: Number; LOCATION: Program).

2. Find the number in the memory space indicated on the left of the equals. (ACTION: Find; OBJECT: Number; LOCATION: Memory).

3. Erase the number in that memory space. (ACTION: Destroy; OBJECT: Number; LOCATION: Memory).

4. Write the new number in that memory space. (ACTION: Create; OBJECT: Number; LOCATION: Memory).

5. Go on to the next statement. (ACTION: Move; OBJECT: Pointer; LOCATION: Program).

6. Do what it says. (ACTION: Allow; OBJECT: Command; LOCATION: Program).

Thus, there is a general structure for each transaction; you can expect some action to be carried out on some object in some location in the computer. The two techniques cited in previous sections can be applied to teaching a transaction-type analysis of statements. It may be noted that statements with the same name may actually consist of different actions. For example, a "Counter Set LET" such as above is different from an "Arithmetic LET" such as LET X = 5/2. Explicit naming and describing of different types of statements with the same keyword may become a part of computer instruction.

4.2 Understanding a Program

What do experts know about computer programming that beginners do not know? One answer is that experts possess much more information and that the information is organized more efficiently. For example, a review of research on teaching people how to become better problem solvers concludes that good problem solving requires that the user has domain-specific knowledge: "All problem solving is based on knowledge" (GREE80). Similarly, Simon (SIM080) estimates that a person needs 50,000 chunks of domain-specific information to become an expert in some domain.

In a classic study, subjects were asked to view briefly presented chess board configurations and then try to reconstruct them (CHAS73). Chess masters performed much better than less experienced players on reconstructing board configurations if the board positions came from actual games; however, the advantage was lost when random board patterns were presented. This finding suggests that experts in chess do not necessarily have better memories, but rather they have a repertoire of many meaningful patterns of board positions. They can chunk several pieces together into one meaningful pattern while a less experienced player must try to remember each piece separately. In an analogous study reported by Shneiderman (SHNE80), experienced and inexperienced programmers were given programs to study. Experienced programmers were able to recall many more lines of code than inexperienced programmers when the program was a meaningful running program; however, when the program consisted of random lines of code the two groups performed at similar levels. Apparently, the experts were able to chunk lines of code together into chunks while less experienced users were less able to form such chunks.

For example, Atwood & Ramsey (ATWO78) suggest that experienced programmers encode a segment such as,

```
SUM = 0
DO 1 I = 1, N
     SUM = SUM + (I)
1 CONTINUE
```

as "CALCULATE THE SUM OF ARRAY X." An experienced programmer has a "schema" for this task and is able to generate a variety of lines of code to accomplish it. In order to provide a more precise description of the "schemas" that are involved in understanding programs, Atwood & Ramsey (ATWO80) used a modified version of Kintsch's (KINT74) propositional analysis. Each statement in the program can be written as a predicate with arguments, and a macrostructure can be constructed. Although a detailed description of Atwood & Ramsey's system is beyond the scope of this paper, their work is promising in that it suggests that knowledge can be represented precisely.

One implication of this work is that it might be possible to explicitly teach the major "chunks" or "schemas" involved in computer programming using the techniques cited in previous sections. Explicit naming and teaching of basic schemas such as these may become part of computer programming curricula.

SUMMARY

This paper is concerned with how to make computers and computer programming more understandable for novices. Two instructional techniques from educational and cognitive psychology are described--using concrete models to represent the computer system, and encouraging the learner to describe technical information in his own words. A review of the effectiveness of these techniques revealed that, under certain conditions, both may enhance the learner's understanding as measured by ability to solve transfer problems. Finally, two major objectives of computing instruction were suggested--enhancing the novice's ability to understand statements and to understand programs.

ACKNOWLEDGEMENTS

I wish to thank Tom Moran for his editoral comments. A shorter version of this paper was presented at the NSF Conference on National Computer Literacy Goals for 1985, held in Reston, Virginia on December 18-20, 1980. Preparation of this paper was supported by Grant NIE-G-0118 from the National Institute of Education.

FOOTNOTES

1. Transfer problems are problems that are different from those given in the text, but can be solved using information in the text. Since the text gave information about how to generate single statements and simple programs, these two kinds of problems are not transfer problems. Since the text did not explicitly mention looping, problems that require the generation of a looping program are transfer problems. Similarly, since the text did not explicitly deal with interpretation of programs, interpretation problems are transfer problems in this study. However, looping-interpretation may require much more transfer than the others, since it is most different from the text.

2. These tables are broken down by problem complexity, with more complex problems requiring transfer. The same general pattern is found for both generation and interpretation problems. Table 13 shows data for interpretation problems only, in order to avoid unnecessary complexity. However, this table cannot be directly compared with Table 11.

REFERENCES

ANDE77 Anderson, R.C. The notion of schemata and the educational enterprise. In R.C. Anderson, R.J. Spiro, & W. E. Montague (Eds.), Schooling and the acquistion of knowledge. Hillsdale, N.J.: Erlbaum, 1977.

ATWO78 Atwood, M.E. & Ramsey, H.R. Cognitive structure in the comprehension and memory of computer programs: An investigation of computer programming debugging. (ARI Technical Report TR-78-A210.) Englewood, Colorado: Science Applications, Inc., August, 1978.

AUSU60 Ausubel, D.P. The use of advance organizers in the learning and retention of meaningful verbal material. Journal of Educational Psychology, 1960, 51, 267-272.

AUSU63 Ausubel, D.P. The psychology of meaningful verbal learning. New York: Gruene and Stratton, 1963.

AUSU68 Ausubel, D.P. Educational psychology: A cognitive view. New York: Holt, Rinehart & Winston, 1968.

AUSU62 Ausubel, D.P. & Fitzgerald, D. Organizer, general background, and antecedent learning variables in sequential verbal learning. Journal of Educational Psychology, 1962, 53, 243-249.

AUSU77 Ausubel, D.P., Novak, J.D., & Hanesian, R. Educational psychology: A cognitive view, Second Edition. New York: Harper & Row, 1977.

AUSU61 Ausubel, D.P. & Fitzgerald, D. The role of discriminability in meaningful verbal learning and retention. Journal of Educational Psychology, 1961, 52, 266-274.

BARN75 Barnes, B.R. & Clawson, E.U. Do advance organizers facilitate learning? Recommendations for further research based on an analysis of 32 studies. Review of Educational Research, 1975, 45, 637-659.

BART32 Bartlett, F.C. Remembering. Cambridge: Cambridge University Press, 1932.

BOWE72 Bower, G.H. Mental imagery and associating learning, In L. Gregg (Ed.), Cognition in Learning and Memory. New York: Wiley, 1972.

BRAN79 Bransford, J.D. Human cognition. Monterey, CA: Wadsworth, 1979.

BRAN72 Bransford, J.D. & Johnson, M.K. Contextual prerequisites for understanding: Some investigations of comprehension and recall. Journal of Verbal Learning and Verbal Behavior, 1972, 11, 717-726.

BROW49 Brownell, W.A. & Moser, H.E. Meaningful vs. mechanical learning: A study in grade III subtraction. In Duke University research studies in education, No. 8. Durham, N.C.: Duke University Press, 1949, 1-207.

CHAS73 Chase, W.G. & Simon, H.A. Perception in chess. Cognitive Psychology, 1973, 4, 55-81.

DANS78 Dansereau, D. The development of a learning strategies curriculum. In H.F. O'Neal (Ed.), Learning strategies. New York: Academic Press, 1978.

DOOL71 Dooling, D.J. & Lachman, R. Effects of comprehension on the retention of prose. Journal of Experimental Psychology, 1971, 88, 216-222.

DOOL72 Dooling, D.J. & Mullet, R.L. Locus of thematic effects on retention of prose. Journal of Experimental Psychology, 1973, 97, 404-406.

DUBO76 du Boulay, B. & O'Shea, T. How to work the LOGO machine. University of Edinburgh: Department of Artificial Intelligence, Paper No. 4, 1976.

DUBO80 du Boulay, B., O'Shea, T. & Monk, J. The black box inside the glass box: Presenting computering concepts to novices. University of Edinburgh: Department of Artifical Intelligence, Paper No. 133, 1980.

EWER32 Ewert, P.H. & Lambert, J.F. The effect of verbal instructions upon the formation of a concept, Journal of General Psychology, 1932, 6, 400-411.

FITZ63 Fitzgerald, D. & Ausubel, D.P. Cognitive versus affective factors in the learning and retention of controversial material. Journal of Educational Psychology, 1963, 54, 73-84.

GAGN62 Gagne, R.M. & Smith, E.C. A study of the effects of verbalization on problem solving. Journal of Experimental Psychology, 1962, 63, 12-18.

GOUL74 Gould, J.D. & Ascher, R.M. Query by non-programmers. IBM Research Report, Yorktown Hts., NY, 1974.

GREE76 Greeno, J.G. Cognitive objectives of instruction. In D.K. Dahr (Ed.), Cognition and instruction. Hillsdale, NJ: Erlbaum, 1976.

GREE80 Greeno, J.G. Trends in the theory of knowledge for problem solving. In D.T. Tuma & F. Reif (Eds.), Problem solving and education: Issues in teaching and research. Hillsdale, NJ: Erlbaum, 1980.

GROT68 Grotelueschen, A. & Sjogren, D.D. Effects of differentially structured introductory materials and learning tasks on learning and transfer. American Educational Research Journal, 1968, 5 277-202.

KATO42 Katona, G. Organizing and memorizing. New York: Columbia University Press, 1942.

KINT74 Kintsch W. The representation of meaning in memory. Hillsdale, NJ: Erlbaum, 1974.

KOHL25 Kohler, W. The mentality of apes. New York: Liveright, 1925.

LAWT77 Lawton, J.T. & Wanska, S.K. Advance organizers as a teaching strategy: A reply to Barnes and Clawson. Review of Educational Research, 1977, 47, 233-244.

LESH76 Lesh, R.A. The influence of an advance organizer on two types of instructional units about finite geometrics, Journal of Research in Mathematics Education, 1976, 7, 82-86.

MAYE75a Mayer, R.E. Information processing variables in learning to solve problems. Review of Educational Research, 1975, 45, 525-541.

MAYE75b Mayer, R.E. Different problem-solving competencies established in learning computer programming with and without meaningful models. Journal of Educational Psychology, 1975, 67, 725-734.

MAYE76 Mayer, R.E. Some conditions of meaningful learning for computer programming: Advance organizers and subject control of frame order. Journal of Educational Psychology, 1976, 143-150.

MAYE77 Mayer, R.E. Different rule system for counting behavior acquired in meaningful and rote contexts of learning. Journal of Educational Psychology, 1977, 69, 537-546.

MAYE78 Mayer, R.E. Advance organizers that compensate for the organization of text. Journal of Educational Psychology, 1978, 70, 880-886.

MAYE79a Mayer, R.E. Can advance organizers influence meaningful learning? Review of Educational Research, 1979, 49, 371-383. (a)

MAYE79b Mayer, R.E. Twenty years of research on advance organizers: Assimilation theory is still the best predictor of results. Instructional Science, 1979, 8, 133-167. (b)

MAYE79c Mayer, R.E. A psychology of learning BASIC. Communications of the ACM, 1979, 22, 589-594.

MAYE80a Mayer, R.E. Elaboration techniques for technical text: An experimental test of the learning strategy hypothesis. Journal of Educational Psychology, 1980, 72, in press. (a)

MAYE80b Mayer, R.E. & Bromage, B. Different recall protocols for technical text due to sequenching of advance organizers. Journal of Educational Psychology, 1980, 72, in press.

MAYE80c Mayer, R.E. & Cook. L. Effects of shadowing on prose comprehension and problem solving. Memory and Cognition, 1980, 8, in press.

MERR66 Merrill, M.D. & Stolurow, L.M. Hierarchical preview vs. problem oriented review in learning in imaginary science. American Educational Research Journal, 1966, 3, 251-261.

MINS75 Minsky, M. A framework for representing knowledge. In P. Winston (Ed.), The psychology of computer vision, New York: McGraw-Hill, 1975.

ONEI78 O'Neil, H.F. Learning strategies. New York: Academic Press, 1978.

PAIV69 Paivio, A. Mental imagery in associative learning and memory. Psychological Review, 1969, 76, 241-263.

PEPE78 Peper, R.J. & Mayer, R.E. Note taking as a generative activity. Journal of Educational Psychology, 1978, 70, 514-522.

RAYE73 Raye, C. Considerations of some problems of comprehension. In W.G. Chase (Ed.), Visual information procressing. New York: Academic Press, 1973.

REIS77 Reisner, P. Use of psychological experimentation as an aid to development of a query language. IEEE Transactions on Software Engineering, 1977, 3, 218-229.

RESN80 Resnick, L.B. & Ford, S. The psychology of mathematics learning. Hillsdale, NJ: Erlbaum, 1980.

RING71 Ring, D.G. & Novak, J.D. Effects of cognitive structure variables on achievement in college chemistry. Journal of Research in Science Education, 1971, 8, 325-338.

ROYE75 Royer, J.M. & Cable, G.W. Facilitated learning in connected discourse. Journal of Educational Psychology, 1975, 67, 116-123.

ROYE76 Royer, J.M. & Cable, G.W. Illustrations, analogies, and facilitative transfer in prose learning. Journal of Educational Psychology, 1976, 68, 205-209.

RUME75 Rumelhart, D.E. Notes on a schema for stress. In D.G. Bobrow & A. Collins (Eds.), Representation and understanding. New York: Academic Press, 1975.

SCAN67 Scandura, J.M. & Wells, J.N. Advance organizers in learning abstract mathematics. American Educational Research Journal, 1967, 4, 295-301.

SCHA77 Schank, R.C. & Abelson, R.P. Scripts, plans, goals and understanding. New York: Wiley, 1977.

SCHU75 Schumacher, G.M., Liebert D, & Fass, W. Textural organization, advance organizers, and the retnetion of prose material. Journal of Reading Behavior, 1975, 7, 173-180.

SHNE80 Shneiderman, B. Software psychology: Human factors in computer and information systems. New York: Winthrop, 1980.

SIMO80 Simon, H.A. Problem solving and education. In D.T. Tuma & F. Reif (Eds.), Problem solving and education: Issues in teaching and research. Hillsdale, NJ: Erlbaum, 1980.

SMIT69 Smith, R.J. & Hesse, K.D. The effects of prereading assistance on the comprehension and attitudes of good and poor readers. Research on the Teaching of English, 1969, 3, 166-167.

THOR77 Thorndyke, P.W. Cognitive structures in comprehension and memory of narrative discourse. Cognitive Psychology, 1977, 9, 77-110.

WEAV72 Weaver, F., & Suydam, M. Meaningful instruction in mathematics education. Mathematics Education Reports, 1972.

WEIN78 Weinstein, C. Elaboration skills as a learning strategy. In H.F. O'Neil (Ed.), Learning strategies. New York: Harper & Row, 1978.

WERT59 Wertheimer, M. Productive thinking. New York: Harper & Row, 1978.

WEST76 West, L.H.T. & Fensham, D.J. Prior knowledge or advance organizers as effective variables in chemistry learning. Journal of Research in Science Teaching, 1976, 13, 297-306.

WHIT80 White, R.T. & Mayer, R.E. Understanding intellectual skills. Instructional Science, 1980, 9, 101-127.

WITT74 Wittrock, M.C. Learning as a generative process. Educational Psychologists, 1974, 11, 87-95.

Table 1

Seven Statements Used in BASIC-like Instructional Booklet

Name	Example
READ	P1 READ (A1)
WRITE	P2 WRITE (A1)
EQUALS	P3 A1 = 88
CALCULATE	P4 A1 = A1 + 12
GOTO	P6 GO TO P1
IF	P5 IF (A1 = 100) GO TO P9
STOP	P9 STOP

Table 2

Examples of Six Types of Test Problems for a BASIC-like Language

Generation-Statement	Interpretation-Statement
Given a number in memory space A5, write a statement to change that number to zero.	A5 = 0

Generation-Nonloop	Interpretation-Nonloop
Given a card with a number on it is input, write a program to print out its square.	P1 READ (A1) P2 A1 = A1 * A1 P3 WRITE (A1) P4 STOP

Generation-Looping	Interpretation-Looping
Given a pile of data cards is input, write a program to print out each number and stop when it gets to card with 88 on it.	P1 READ (A1) P2 IF(A1 = 88) GO TO P5 P3 WRITE (A1) P4 GO TO P1 P5 STOP

Table 3

Proportion of Correct Answers on Transfer Test by Type of Problem for Model and Control Groups

	Generation			Interpretation		
	Statement	Nonloop	Looping	Statement	Nonloop	Looping
Model Group	.63	.37	.30	.62	.62	.09
Control Group	.67	.52	.12	.42	.32	.12

Note. 20 subjects per group; interaction between group and problem type, $p < .05$.

Table 4

Proportion of Correct Answers on Transfer Test by Type of Problem for Before and After Groups

	Generation			Interpretation		
	Statement	Nonloop	Looping	Statement	Nonloop	Looping
Before	.57	.50	.20	.47	.63	.17
After	.77	.63	.13	.27	.40	.17

Note. 20 subjects per group; interaction between group and problem type, $p < .05$.

Table 5

Example of Conceptual, Format, and Technical Idea Units

Type	Idea Unit
Technical	READ is one kind of statement.
Format	The format is READ ( ).
Format	An address name goes in the parenthesis.
Conceptual	An address name is a space in the computer's memory.
Conceptual	There are 8 memory spaces.
Technical	The spaces are called A1, A2 ....
Technical	An example is, READ (A2).
Conceptual	First, the computer checks the number from the top data card.
Conceptual	Then, that number is stored in space A2.
Conceptual	The previous number in A2 is destroyed.
Conceptual	Then the data card is sent out of the computer.
Conceptual	This reduces the pile of data card by 1.
Conceptual	Then, go on to the next statements.

Table 6

Average Number of Recalled Idea Units for the Before and After Groups

	Idea Units			Intrusions		
	Technical	Format	Conceptual	Inappropriate	Appropriate	Model
	(14)	(12)	(35)			
Before	5.0	1.9	6.6	1.5	1.3	3.1
After	6.0	2.9	4.9	2.5	.8	.5

Note. 30 subjects per group; interaction between group and problem type, $p < .05$.

Numbers in parentheses indicate total possible.

Table 7

Eight Statements Used in File Management Language Booklet

Name	Example
FROM	FROM AUTOMOBILE
FOR	FOR WEIGHT IS CALLED 3000 OR MORE
AND FOR	AND FOR COLOR IS CALLED GREEN
OR FOR	OR FOR MAKE IS CALLED FORD
LIST	LIST NAME
COUNT	COUNT
TOTAL	TOTAL CURRENT VALUE
LET	LET TOTAL ÷ COUNT BE CALLED AVERAGE

Table 8

Examples of Test Problems for a File Management Language

Sort 1

List the owners' names for all cars weighing 3000 pounds or more.

```
FROM AUTOMOBILE
FOR WEIGHT IS CALLED 3000 OR MORE
LIST NAME
```

Sort 2

List the owners' names for all late model green Fords.

```
FROM AUTOMOBILE
FOR YEAR IS CALLED 1976 OR MORE
AND FOR COLOR IS CALLED GREEN
AND FOR MAKE IS CALLED FORD
LIST NAME
```

Count

How many cars are registered in Santa Barbara County?

```
FROM AUTOMOBILE
FOR HOME COUNTY IS CALLED SANTA BARBARA
COUNT
LIST COUNT
```

Compute 1

What is the average current value of all cards?

```
FROM AUTOMOBILE
COUNT
TOTAL CURRENT VALUE
LET TOTAL ÷ COUNT BE CALLED AVERAGE
LIST AVERAGE
```

Compute 2

What percentage of 1977 cars are Chevrolets?

```
FROM AUTOMOBILE
FOR YEAR IS CALLED 1977
COUNT
LET THIS BE CALLED COUNT 1
AND FOR MAKE IS CALLED CHEVROLET
COUNT
LET THIS BE CALLED COUNT 2
LET COUNT 2 ÷ COUNT 1 BE CALLED AVERAGE
LIST AVERAGE
```

Table 9

Proportion of Correct Answers on Transfer Test for Model and Control Groups--File Management Language

	Type of Test Problem				
	Sort-1	Sort-2	Count	Computer-1	Compute-2
Model Group	.66	.66	.63	.58	.45
Control Group	.63	.44	.43	.33	.22

Note. 20 subjects per group; group x problem type interaction, $p < .07$.

Table 10

Example of the Model Elaboration Exercise in the Programming Text

Model Elaboration

Consider the following situation. An office clerk has an in-basket, a save basket, a discard basket, and a sorting area on the desk. The in-basket is full of records. Each one can be examined individually in the sorting area of the desk and then placed in either the same or discard basket. Describe the FOR statement in terms of what opertations the clerk would perform using the in-basket, discard basket, save basket, and sorting area.

Table 11

Proportion of Correct Answers on Transfer Test by Type of Problem for Model Elaboration and Control Groups

	Type of Test Problem				
	Sort-1	Sort-2	Count	Compute-1	Computer-2
Model Elaboration Group	.65	.58	.64	.64	.45
Control Group	.66	.64	.41	.38	.27

Note. 20 subjects per group; group x problem type interaction, $p < .05$.

Table 12

Example of the Comparative Elaboration Exercise in the Programming Text

Comparative Elaboration

How is the FOR command like the FROM command?

How is the FOR command different than the FROM command?

Table 13

Proportion Correct on Transfer Test for Comparative Elaboration and Control Groups

	Type of Problem				
	Sort-1	Sort-2	Count	Compute-1	Compute-2
Comparative Elaboration	.90	.90	1.00	.75	.55
Control	.90	.90	.65	.65	.25

Note. Data is for interpretation problems only. 13 subjects per group.

group x problem type interaction, $p < .05$.

Table 14

Average Number of Recalled Idea Units for Model Elaboration, Comparative Elaboration and Control Groups

	Type of Idea Units	
	Technical	Conceptual
	(19)	(52)
Model Elaboration	5.3	13.9
Comparative Elaboration	9.4	14.1
Control	7.5	7.5

Note. 20 subjects per group; group x type interaction, $p < .05$, for low ability subjects. Numbers in parenthesis indicate total possible.

Table 15

Proportion of Correct Answers on Transfer Test for Notes and No-Notes Groups

	Problem Type	
	Generative	Interpretive
Low Ability Subjects		
Notes Group	.39	.56
No-Notes Group	.49	.33
High Ability Subjects		
Notes Group	.67	.62
No-Notes Group	.60	.60

Note. 15 subjects per group; effect of ability, $p < .01$; interaction between group ability, and problem type, $p < .025$.

Table 16

Average Number of Recalled Idea Units for Notes and No-Notes Groups

	Type of Idea Units		
	Technical	Conceptual	Intrusions
	(28)	(36)	
Notes Group	10.4	7.2	3.9
No-Notes Group	9.4	4.7	2.4

Note. 20 subjects per group; interaction between group and type of recall, $p < .025$. Numbers in parentheses indicate total possible.

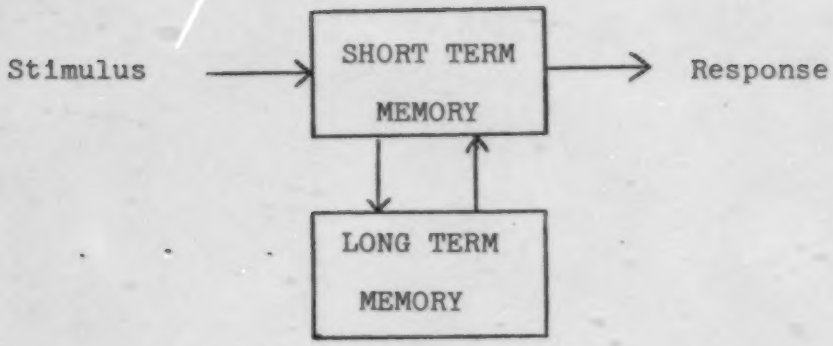


Figure 1. Some information processing components of meaningful learning. Condition a is transfer of new information from outside to short term memory. Condition b is availability of assimilative context in long term memory. Condition c is activation and transfer of old knowledge from long term memory to short term memory.

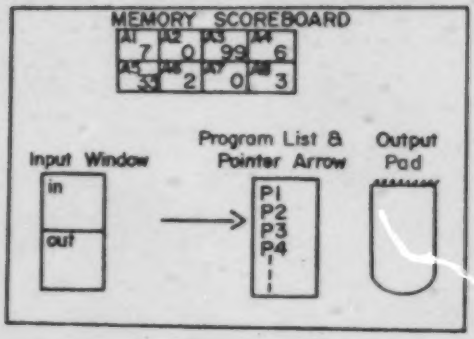


Figure 2. A concrete model of the computer for a BASIC-like language.

Figure 2a. Description for Model

The figure above represents a simple computer system which you will learn about in this experiment. The computer is made-up of three main parts: (1) INPUT & OUTPUT WINDOWS which allow communication between the computer's memory and the outside world, (2) MEMORY SCOREBOARD which stores information in the computer, and (3) PROGRAM LIST & POINTER ARROW which tell the computer what to do and what order to go in. Each of these three parts will now be explained.

INPUT & OUTPUT WINDOW

Notice that to the far left is an input window divided into two parts. A pile of computer cards with numbers punched into them can be put in the left part of the window; as the computer finishes processing each card it puts the card on the right side of the input window. Thus when the computer needs to find the next data card, it takes the top card on the left side of the input window; when it is done with the card, it puts it on the right side.

On the far right is the output window. This is where printed messages (in this case, only numbers can be printed) from the computer's memory to the outside world appear. Each line on the printout is a new message (i.e., a new number).

Thus the computer can store in memory a number that is on a card entered through the input window or it can print out what it has in memory onto a printout at the output window. The statements which put the input and output windows to work are READ and WRITE statements, and each will be explained later on.

MEMORY SCOREBOARD

Inside the computer is a large scoreboard called MEMORY. Notice that it is divided into eight spaces with room for one score (one number) in each space.

Figure 2a (continued)

Also notice that each space is labeled with a name -- A1, A2, A3, A4, A5, A6, A7, A8. These labels or names for each space are called "addresses" and each of the eight addresses always has some number indicated in its space. For example, right now in our figure, A1 shows a score of 81, A2 has the number 17, etc.

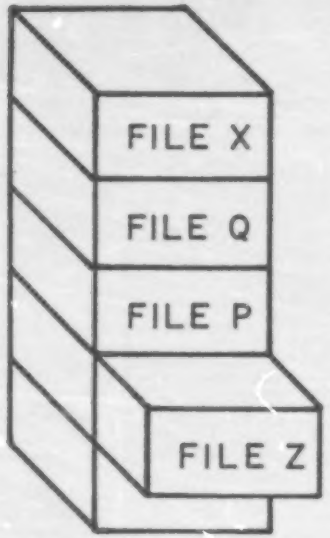
It is possible to change the score in any of the eight spaces; for example, the score in box A1 can be changed to 0, and you will learn how to change scores in memory later on when we discuss EQUALS statements and CALCULATION statements.

PROGRAM LIST & POINTER ARROW

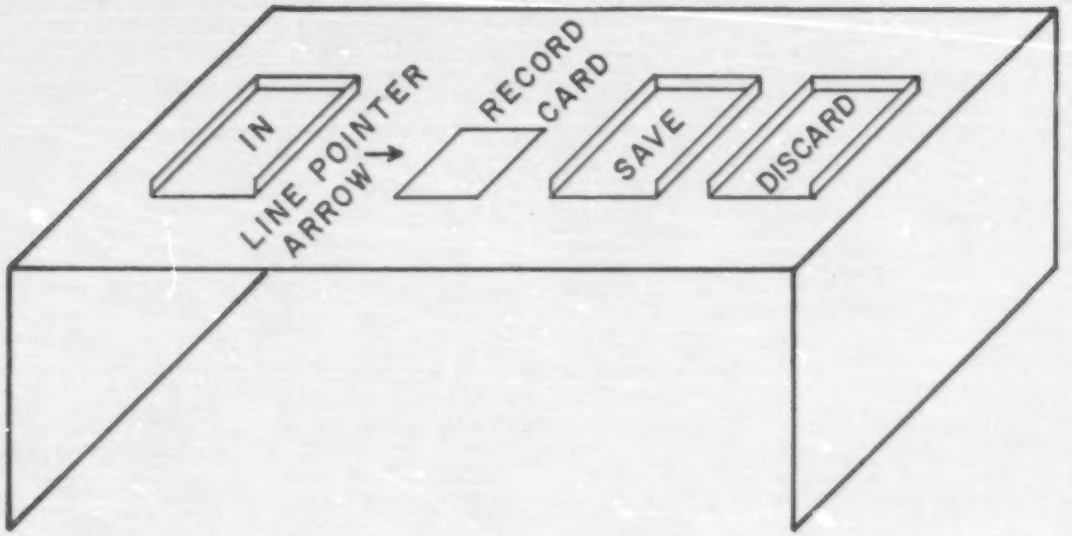
Inside the computer to the right of the MEMORY is a place to put a list of things to do called PROGRAM LIST and an arrow which indicates what step in the list the computer should work on.

Notice that each line in the PROGRAM LIST has a number so that the first line is called P1, the second step is P2 and so on. When a program is inserted in the step indicator arrow will point to the first line (P1); when the first step is finished the arrow will go to the next step on the list (P2), and so on down the list. You will learn how to control the order of steps later on when the IF statement, GO TO statement and STOP statement are discussed.

FILE CABINET



SORTING BASKETS



MEMORY SCOREBOARD

COUNT 55	TOTAL 212	AVERAGE 3
COUNT1 12	TOTAL1 0	AVERAGE1 0
COUNT2 7	TOTAL2 714	AVERAGE2 102
COUNT3 33	TOTAL3 33	AVERAGE3 1
COUNT4 3	TOTAL4 150	AVERAGE4 50

OUTPUT PAD

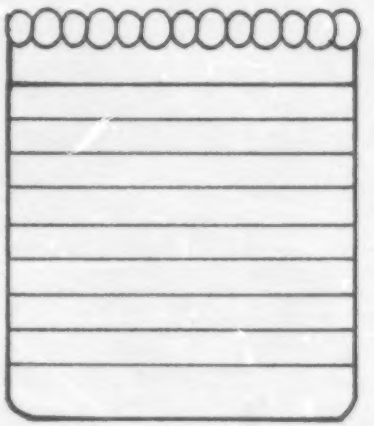


Figure 3

Figure 3a. Description for Model

The computer is capable of three main functions: sorting record cards into sorting baskets, remembering numbers on its memory scoreborad, and outputing information to the world through its message pad.

To understand the sorting function of the computer you could think of an office worker sitting at a desk with three sorting baskets, a line pointer arrow, and file cabinet with many drawers. Each drawer of the file cabinet contains a different set of records; the name of the file is indicated on each drawer. If the worker needs all the records in a particular file all the worker needs to do is open that drawer and take out all the records. To avoid mix-ups the clerk can take out all the records of only one file at a time; if the clerk needs to bring records from a certain file drawer to his desk, first all the records from all other files must be put back in their proper drawers. Thus, a worker may have all the records for only one file on his desk at a time. These could be placed in the "in basket" which is on the left side of the clerk's desk--it thus contains all of the to-be-processed record cards, waiting for the office clerk to look at them. In the middle of the desk is a work area with a line pointer arrow; the clerk may place only one card in the work area at a time, and the pointer arrow points to just one line at a time. To the right are two more baskets--the "save basket" and the "discard basket." If a record card passes the clerk's inspection it is placed on top of the pile of cards in the "save basket"; but if it fails it is placed in the top of the pile of cards in the "discard basket." The procedure the office worker uses is to take the top card from the "in basket", place it in the work area with a pointer arrow aimed at one line, and based on inspection of this line to move that card to either the "save" or "discard basket." The worker continues until

all of the the records in the "in basket" have been processes so that the "in basket" is empty and the "save" and "discard baskets" contain all the records; then, sometimes the worker might be asked to take the pile in either the "save" or the "discard basket" and put them in the "in basket" for further processing.

To understand the memory function of the computer, think of a memory scoreboard. The scoreboard consists of 15 rectangular spaces like a classroom blackboard divided into 15 spaces. Each space has a lable such as COUNT2, and each space has one number (of any length) in it. The office worker may count all the records that have been stored in the SAVE Basket, and this number could be stored in one of the spaces on the scoreboard. When a new number is stored in a space on the scoreboard, the old number is erased. However, when the office worker copies a number from one of the memory spaces onto the output pad the number is not erased.

To understand the output function of the computer think of a telephone message pad. To communicate with the outside world the computer can write one piece of information on each line of the pad. It is fills all the lines on one page, it will just turn to the next page and begin with the top line. The office worker may write down two kinds of information on the output pad: a number may be copied from one of the spaces on the scoreboard onto the pad (but this does not alter the number on the scoreboard), or information that is on each card in the Save Basket can be copied onto the output pad.

CHAPTER 3

TASK ANALYSIS, DEVELOPMENT OF EVALUATION INSTRUMENTS, AND DIAGNOSIS OF BUGS FOR CALCULATOR LANGUAGE

Note

This article has been published as the following citation:

Mayer, R. E., & Bayman, P. Analysis of student's intuitions about the operation of electronic calculators. Technical Report No. 80-4. Santa Barbara: University of California, 1980.

This article has been submitted for publication as the following citation:

Mayer, R. E., & Bayman, P. Users' misconceptions concerning the operation of electronic calculators. Cognition & Instruction, under review.

This research was also presented at a professional meeting as the following:

Mayer, R. E., & Bayman, P. Analysis of students' intuitions about the operation of electronic calculators. American Educational Research Association, Los Angeles, April 13-17, 1981.

Abstract

Thirty-three novice and 33 expert users predict what number will be in the display of a calculator after a sequence of key presses (such as 2 + 3 +). The performance of each subject on 88 problems is formally described as a 13 line production system. Conditions are key presses (such as + after a number); actions are changes in the display or internal registers (such as display the evaluation of the expression in the register). Large individual differences are observed. Differences among subjects include when an expression is evaluated and displayed (e.g., after a + key, x key, = key, and/or number key), whether or not the display is incremented when two operation keys are pressed in sequence (e.g., 2 + + or 2 x x), whether or not the display is incremented when an equals is pressed after an operation (e.g., 2 + =), what the order of arithmetic will be in a chain (e.g., 2 + 3 x 7). Experts are more consistent in their performance and tend to be more likely than novices to base answers on standard operating systems. Implications for developing a theory of computer literacy are discussed.

Users' Misconceptions Concerning the Operation of Electronic Calculators

Within the past decade electronic calculators have become a part of our society, including widespread and rapid acceptance in schools (Mullish, 1976). Based on a survey of articles and editorials published within the past few years in Arithmetic Teacher and Mathematics Teacher, as well as a policy statement by the National Council of Teachers of Mathematics (1976), it is clear that calculators will play an important role in the education of our children. For example, the following statements by mathematics educators are typical; "All students will profit from having access to a calculator" (Gawronski & Coblentz, 1976). "I propose that we make fullest possible use of calculators in all grades of our school" (Hopkins, 1976). "Not since the printing press has any invention had such potential for revoluntionizing education, particularly mathematics education" (Rudnick & Krulik, 1976).

However, in spite of these optimistic predictions and endorsements, the research community has been slow in providing information that might be useful in this impending calculator-curriculum revolution. For example, most experimental studies to date have compared changes in achievement and/or attitude scores for students who use calculators in school versus students who were not allowed to use calculators (Gaslin, 1975; Roberts & Fabrey, 1978; Roberts & Glynn, 1979; Schnur & Lang, 1976; Suydam, Note 1). In a recent review of 34 studies, most of which were not published in journals, Roberts (1980) observed that there was clear evidence that calculators improve computational efficiency but no consensus concerning effects on higher level conceptual achievement or attitudes towards mathematics. Thus, he concludes that "the research literature offers no guidance" concerning how to incorporate calculators into school curricula.

The present paper does not attempt to address the question of whether calculators influence changes in mathematics achievement and attitude scores. Rather, this paper is based on the idea that since calculators will become a part of everyday life for students, it is important to know how people come to understand and interact with calculators. The calculator represents a student's first introduction to a computer, to a computer language, and to computer literacy in general. Pressing a key is analogous to a computer command. In spite of tremendous breakthroughs in development of improved calculator hardware for the mass market, there has been comparatively little work on what Shneiderman (1980) calls "software psychology". That is to say, we know very little about how people understand calculators, what types of instruction will help people become creative users of calculators, why some people seem to not use them very well, or how to design operating systems that make psychological sense.

Since calculator useage seems so simple and since even children can teach themselves to use a calculator in a short time, some educators have suggested that explicit instruction or concern about users' understanding of calculators is not needed (Bell, 1976). This might be correct if one's goal is simply to have students use the calculator as a "black box" that gives answers for mundane computation. However, when the goal is to promote productive problem solvers, there is reason to believe that the student's understanding of how the calculator operates is important. For example, Scandura, Lowerre, Veneski & Scandura (1976) found that some students who are left to teach themselves develop bizzare intuitions; for example, some subjects concluded that the plus (+) and equals (=) key did nothing since they caused no visible change in the display. On the other hand, Meyer (1981) found that by letting fourth-graders explore the functions of the operation keys some of them discovered that pushing the same

operation key more than once would cause the calculators to repeat the process with the number last entered. Leaving the understanding of the basic functions of the calculator to chance might create as many students with incorrect intuitions as correct ones. Thus, although two users may be able to use the calculator to solve basic computational problems, there may be large individual differences in the way users understand and interact with calculators.

Elucidation of individual differences in students' intuitions about the operation of calculators (i.e., students' conception of what goes on inside the "black box" when a key is pressed) is the logical first step in building a theory of computer literacy, and is the goal of the present study. In particular, this study addresses two related issues.

(1) What knowledge do people have about how calculators work? Since most users are "self-taught" and seem to be able to use their calculators, an important issue concerns what they have learned. Since some intuitions may lead to more creative use of calculators and to better transfer to computer languages (such as programmable calculators or BASIC) it would be useful to be able to describe exactly what people's intuitions are. Recent research on the cognitive analysis of computational skill suggests that two children with the same apparent performance may be using entirely different conceptions of computation. For example, Groen & Parkman (1972) have developed cognitive models of addition, and Woods, Resnick & Groen (1975) have developed models of subtraction. More recently, Brown & Burton (1975) have developed a BUGGY program which serves to diagnose problems in a child's arithmetic procedures by developing a formal description of the algorithm that the child is using. Successful application of cognitive analysis tools to the problem of describing computational skill encourages the idea that similar techniques can be used to formalize how students understand calculator logic.

(2) What are the differences in intuitions among individuals? For example, do "experts" have different intuitions than "novices?" In recent studies, Larkin (1979) and Simon & Simon (1978) have been able to formalize differences between what experts and novices know about solving physics problems. Larkin (1979) has been able to represent the differences in terms of differences in the organization and size of productions in a production system. Similar techniques may be applied to representing differences between experts and novices in the domain of calculator use.

STUDIES 1 AND 2

The purpose of study 1 was to determine the knowledge that ordinary users have concerning the operation of hand-held calculators. In particular, the goal was to formally describe each subject's conception of the calculator's operating system. The purpose of study 2 was to determine whether the formal descriptions developed in study 1 would also describe the conceptions of people who were more knowledgeable about operating systems. For purposes of this paper, subjects in study 1 are labeled "novices" and subjects in study 2 are labeled "experts".

Method

Subjects

The subjects in study 1 were 46 college undergraduates recruited from the subject pools at the University of Pittsburgh and the University of California, Santa Barbara.[1] All subjects participated in order to fulfill a requirement for their introductory psychology courses. Subjects in study 1 had no previous experience with computer programming nor with the concept of operating systems. Thirteen of the 46 subjects in study 1 produced inconsistent performance, so only data for the remaining 33 subjects was used for the analyses. Of these 33 subjects there were 16 females and 17 males, and 26 of the subjects owned a

calculator. The mean age was 18 years, the mean GPA was 3.0, the mean SAT-Quantitative score was 547, the mean SAT-Verbal score was 491.

The subjects in study 2 were 35 college undergraduates recruited from a course in computer programming at the University of California, Santa Barbara.[2] All subjects had taken at least one previous course in computer programming and were currently in a course that included study of operating systems. Of these 35 subjects, one gave inconsistent performance and one failed to follow directions. The remaining 33 subjects were retained for analysis in this study. There were 13 females and 20 males, and 32 of the subjects owned a calculator. The mean age was 21 years, the mean GPA was 2.9, the mean SAT-Quantitative score was 669, and the mean SAT-Verbal score was 552.

The main difference between subjects in study 1 (novices) and study 2 (experts) is that all of the experts had formal instruction in computer programming and some introduction to operating systems while none of the novices did; the experts were older, $t(60) = 2.66$, $p < .01$; and the experts had higher SAT-Quantitative scores, $t(44) = 4.67$, $p < .001$. Thus, while the main focus was on comparing "liberal arts" students who had no formal programming experience to "engineering" students who had formal training in programming, any comparisons between the subjects in the two studies must be made in light of other differences such as age and SAT scores.

Materials

The materials in study 1 and study 2 were essentially identical. Materials consisted of a questionnaire, an instruction sheet, and two two-page problem sets.

The questionnaire was an 8½ x 11 inch sheet of paper which asked the subject to indicate his or her age, sex, GPA, SAT scores, year in school, major

in school, experience with computer programming, experience with calculators, and previous mathematics courses. In particular, subjects were asked to indicate whether they owned or regularly used a particular calculator, and if so, to give the name of the model. In addition, subjects were asked to check a box corresponding to the average number of minutes per week they used a calculator--less than 10, 10 to 30, 30 to 60, more than 60.

The instructions for the problems were typed onto an 8½ x 11 inch sheet of paper. Instructions described a typical calculator and the task.

Each of the two problem sets consisted of 44 problems typed onto two 8½ x 11 inch sheets of paper with one problem per double-spaced line. Each problem presented a series of key strokes and provided a blank space for the subject to indicate what number would be in the display. Each problem contained from one to seven key strokes and each key stroke was either a single digit (2, 3 or 7), a plus key (+), a multiply key (x), or an equals key (=). The two problem sets (Set A and Set B) provided for a reliability check since each problem in Set A corresponded to a problem of the same form in Set B, and both sets presented the corresponding problems in the same order. However, the specific digits used in corresponding problem were different. For example, problems 2+3+7 or 2+3x or 2+=+=+= in set A corresponded to 7+3+2 or 7+3x or 7+=+=+=, respectively, in Set B. The complete list of Set A problems is given in the left side of Table 1.

Procedure

The procedures were essentially identical in study 1 and study 2 except that subjects were run individually in study 1 and were run as a group in study 2.

First, each subject filled out the questionnaire. Then the instructions were presented. Subjects were asked to suppose that they had just been given a new standard four-function calculator that worked efficiently, and to suppose

that they would be using this calculator throughout the session. They were told that for each problem, their job was to predict what number would be in the display of the calculator after the series of key presses (assuming the calculator was cleared at the beginning of the problem). Then, one of the problem sets was randomly selected and given to the subject; when the subject finished the first set, the other set was given. Subjects were asked to put their answers in the space next to each problem and there was no time limit.

Results and Discussion

Scoring

The data for each subject in each study consisted of a number (i.e., the subject's answer) for each of the 88 problems.

Reliability of Performance

Since two forms of the same 44-problem test were administered to each subject, it was possible to determine the reliability of each subject's performance. For each of the possible answers to the 44 problems in set A, corresponding answers were generated for set B. For example, if a subject gave 12 as the answer for 2+3+7 in set A, the corresponding answer for 7+3+2 in set B would also be 12; if a subject gave 7 as the answer for the above problem in set A, the corresponding answer in set B would be 2. Similarly, if a subject gave 2 as the answer for 2+=+=+= in set A, the corresponding answer for 7+=+=+= in set B is 7; if a subject gave 16 for the above problem in set A, the corresponding answer for set B is 56. Reliability scoring was conducted by matching each of the 44 problems in set A with its corresponding problem in set B; if the answers did not correspond, subjects were given a point.

Thirty-three of the 46 subjects in study 1 displayed six or less (i.e., less than 14%) non-matching scores between set A and set B. Data for the 13

subjects who displayed more than six non-matches (i.e., over 14% unreliable answers) were not analyzed further. The mean non-matching score for all 46 subjects in study 1 was 5.3 or 12%; the mean non-matching score for the 33 selected subjects in study 1 was 2.6 or 6%.

In study 2, one of the 35 subjects gave more than six non-matching answers between set A and set B. Data for this subject as well as for one subject who failed to follow directions were not included in subsequent analyses. The mean number of non-matching answers for the 33 selected subjects in study 2 was .8 or 2%.

One question that may be raised at this point is whether the experts and novices differ with respect to the reliability of their performances. The proportion of unreliable novices (13 out of 46) was significantly higher than the proportion of unreliable experts (1 out of 35) as determined by a chi-square test, $\chi^2 = 7.28$, df = 1, p F .01. In addition, for the 33 selected subjects in each study, the novices produced significantly more unreliable answers than the experts as determined by a t-test, $t(64) = 10.59$, $p < .001$. Thus, as might be expected, experts were more consistent in the way they answered problems than were the novices.

All subsequent analyses are based on answers to set A for the 33 subjects in each group.[3]

Performance of Subjects Compared to Performance of Calculators

In this section we address the question of which calculator most closely fits the answers given by the subjects. Of the 33 subjects in study 1, 17 owned simple Texas Instruments (TI) models, three owned Sharp models, one owned a Rockwell model, one owned a Hewlett-Packard HP-21, and eleven either did not own a calculator or could not remember what kind they owned.

Since TI, Sharp and Rockwell were the relevant models[4] owned by subjects in study 1, answers to each of the 44 problems in set A were generated using each of the three brands of calculator. Interestingly, while most of the calculators gave identical answers for most problems, there were different answers produced by at least two of the calculators on 20 of the 44 problems.

A difference score was computed for each subject for each of the three calculator brands. The difference score was based on tallying the number of times that the subject's answer was not identical to the calculator's answer for the 44 problems in set A. Mean difference scores in study 1 were 8.8 (20%) for TI, 20.8 (47%) for Rockwell, and 14.9 (34%) for Sharp. A one-way analysis of variance was conducted on the difference scores with brand of calculator as a within subjects factor. The ANOVA revealed that the difference scores listed above were significantly different from one another, F (2,64) = 42.6, $p < .001$. Supplementary Newman-Keuls tests indicated that the score for TI was significantly better than for Rockwell, but no other differences were significant ($p < .05$).

Of the 33 subjects in study 2, 23 owned Texas Instruments (TI) models, two owned Sharp models, three owned Casio models, three owned Hewlett-Packard, and two either did not own a calculator or could not remember what kind they owned. Thus, as with novices the most frequently owned calculator was TI. However, 11 of the 33 novices did not own or could not remember which calculator they owned, while only 2 of the 33 experts fell into this category. According to a chi-square test, this difference between proportions is significant, $\chi^2 = 6.13$, df = 1, $p < .025$.

The mean difference scores in study 2 were 9.0 (20%) for TI, 18.2 (41%) for Rockwell, and 13.7 (31%) for Sharp. A one-way analysis of variance was

conducted on the difference scores with brand of calculator as a within subjects factor. The ANOVA revealed that the difference scores listed above were significantly different from one another, F (2,64) = 23.74, $p < .001$. Supplementary Newman-Kuels tests revealed that the score for TI was significantly better than for Rockwell or for Sharp, and that the score for Sharp was significantly better than for Rockwell ($p < .05$).

Thus, both in study 1 and in study 2, there is evidence that TI's operating system most closely matches the intuitions of subjects for the 44 problems in the test. In comparing experts and novices, there is no evidence of any differences in which calculator gives the best fit; t-tests revealed no differences between experts and novices with respect to their scores for TI, $t(64) < 1$, for Rockwell, $t(64) = 1.58$, or for Sharp, $t(64) = 1.27$.

Performance of Subjects By Type of Calculator They Own

The previous section suggested that subjects in our sample generated performance that more closely matched the performance of a TI calculator than the other calculators we tested. However, since the TI is the brand of calculator that most subjects in our sample owned, the above results may be mainly due to experience with TI calculators. In order to test this idea, subjects in study 1 were divided into two groups: those who owned a TI calculator (n = 17) and those who do not (n = 16).

The mean difference scores for the TI-owners in study 1 were : TI = 8.0, Rockwell = 20.0, and Sharp = 14.1; the mean difference scores for the non-TI owners were: TI = 9.8, Rockwell = 21.6, Sharp = 15.8. As can be seen, for both TI-owners and non TI-owners, the difference scores are least for TI. An analysis of variance was conducted on the difference-score data with group as a between subjects factor and type of calculator as a within subjects factor.

There was a significant difference between the calculators in how well they matched the performance of all subjects, F (2,60) = 51.16, p <.001, but there was no group x calculator interaction, F (4,60) < 1. Thus, there was no evidence that the non-TI owners were different from TI owners with respect to their performance being best fit by a TI calculator. A separate one-way ANOVA was conducted on the data for the non-owners with type of calculator as a within- subjects factor. The differences in difference scores were significant, F (2,30) = 37.57, p F .001; subsequent Newman-Kuels tests showed that the score for the TI were significantly better than for Sharp or Rockwell among non-TI owners (p <.05).

As an additional analysis, the performance of each subject was labeled as TI-like, Rockwell-like or Sharp-like, based on which of the three calculators produced the lowest difference score for the subject. Of the 33 subjects in study 1, 29 were classified as TI-like and 4 were better matched with other brands. For the TI-owners, 15 were classified as TI-like and 2 were best fit by other brands; for the non-TI owners, 14 were classified as TI-like and 2 were best fit by other brands. A Fisher's Exact test revealed that there was no evidence of any differences among the two groups (TI owners versus non-owners) in the proportion of TI-like subjects.

The performance of the experts in study 2, like the novices in study 1, most closely matches the performance of the TI, but this may be due to the fact that TI is the most prevelant calculator used among the experts. As in study 1, this idea was tested by dividing the experts into those who owned a TI calculator (n = 22) and those who did not (n = 11). The mean difference scores for the TI-owners were: TI = 7.6, Rockwell = 34.4, Sharp = 13.2; the main difference scores for the non-TI owners were: TI = 11.8, Rockwell = 17.4, Sharp

= 14.6. An analysis of variance was conducted on the difference score data with ownership group as a between subjects factor and type of calculator as a within subjects factor. As expected, there was a significant difference among the calculators in how well they matched the performance of all subjects, F (2,60) = 19.21, p <.01; however, as in study 1, there was no interaction between group and calculator, F (4,60) < 1. Thus, as in study 1, there was no evidence that non- TI owners were different from TI owners with respect to their performance being best fit by a TI calculator.

As an additional analysis, the performance of each subject in study 2 was labeled as TI-like, Rockwell-like, or Sharp-like, based on which of the three calculators produced the lowest difference scores for each subject. Of the 33 subjects, 26 were classified as TI-like and 7 were better matched by other brands. For TI-owners, 18 were classified as TI-like and 4 were best fit by other brands; for the non-TI owners, 8 were classified as TI-like and 3 were best fit by other brands. A Fisher's Exact test revealed that there was no evidence of any differences among the two groups (TI owners versus non-owners) in the proportion of TI-like subjects.

This section helps clarify the earlier finding that subjects' performance is most closely matched by TI's operating system. Since this finding seems to be present for both TI-owners and non-owners it may be attributed to the "intuitive appeal" of the TI operating system rather than to users having more experience with TI products. However, it should be pointed out that the correspondence between the calculator's answers and the subjects' answers are far from perfect, even when we choose the best fitting calculator.

Performance of Subjects by Amount of Experience With Calculators

This section explores the issue of whether subjects who differ with respect

to how much they use a calculator also differ with respect to intuitions about the operating system underlying the calculator. In order to address this issue, subjects in study 1 were divided into two groups based on their reported weekly use of calculators: low experience--less than 10 minutes per week (n = 17), and moderate experience--10 or more minutes per week (n = 16).

The mean difference scores for study 1 on each of the three calculators brands was TI = 10.2, Rockwell = 22.2, and Sharp = 16.5 for the low experience group, and TI = 7.4, Rockwell = 19.3, and Sharp = 13.3 for the moderate experience group. As can be seen, TI produces the lowest score (i.e., best fit) for subjects' performance in both groups. An analysis of variance was conducted on the difference score data with experience (low vs. moderate) as a between subjects factor and calculator brand as a within subjects factor. As in previous analyses, there was a significant overall difference among the calculators in how closely they fit the intuitions of the subjects, F (2,62) = 40.62, p <.001, and there was no interaction between group and calculator, F (2,62) < 1. Thus, there was no evidence in study 1 that the superior fit of the TI calculator was influenced by how much experience a subject had.

As in the previous section, each subject was classified as being either TI-like, Rockwell-like, or Sharp-like based on which calculator produced the least number of differences with the subject's actual performance. For the low experience group in study 1, 15 subjects were best fit by TI and 2 were best fit by another calculator; for the moderate experience group 14 were best fit by TI and 2 were best fit by another calculator. A Fisher's Exact test showed that there were no significant differences between low experience and moderate experience groups with respect to the proportion of TI-like subjects. Thus, there is no evidence in this analysis that amount of experience with calculators influence the subjects' intuitions about the operating systems of calculators.

Corresponding analyses were conducted on the data for the experts in study 2. The mean difference scores for study 2 were: TI = 9.5, Rockwell = 18.1, and Sharp = 14.4 for the low experience group, and TI = 8.8, Rockwell = 18.3, and Sharp = 13.5 for the moderate experience group. An ANOVA revealed the expected overall effect, $F(2,52) = 18.92$, $p < .001$; but there was no interaction between experience group and calculator, $F(2,62) < 1$, and thus no evidence that the superior fit of the TI calculator was influenced by how much experience a subject had with calculators. For the low experience group in study 2, 6 of the 8 subjects were best fit by TI while for the moderate experience group, 21 of the 25 subjects were best fit by TI. A Fisher's Exact test showed there was no significant difference with respect to the proportion of TI-like subjects among low and moderate experience subjects in study 2. Thus, for both experts and novices, there is no evidence that experience with calculators influences the subjects' intuitions about the operating system of calculators.

Inter-Subject Consistency in Performance

The foregoing analyses indicates that subjects' performance was closest to that of a TI calculator, and that this pattern was not influenced by whether subjects actually owned a TI calculator nor whether they had experience with using a calculator. However, the foregoing analyses also made clear that subjects' performance could not be adequately described as corresponding to one particular calculator's performance, since the best fitting calculator predicted only about 80% of the answers. In this section, we explore the question of how similar or different the subjects' answers were from subject to subject.

Table 1 gives a list of the 44 problems in set A as well as the answers given by subjects in study 1 and study 2. Each answer that was given by any subject is listed (in parentheses) along with the number of subjects who gave that answer in study 1 and in study 2. As can be seen, for each question there is an answer that occurs most often (i.e., the modal answer) and there may be one or more other answers given by some subjects (i.e., alternative answers). The percentage of subjects' answers that are modal answers, i.e., answers that correspond to the most common answer for each question is 83% for study 1 and 81% for study 2. A t-test revealed that the experts and novices do not differ with respect to the percentage of modal answers, $t(64) < 1$. Kolmogorov-Smirnov's One-Sample tests based on the data for both studies together indicated that the following problems produce significant ($p < .05$) number of non-modal answers: 4, 13, 15, 17, 19, 21, 23, 24, 25, 27, 28, 29 and 31 through 44. Thus, the data in Table 1 encourages the conclusion that there are substantial individual differences among subjects in their intuitions about calculators. In order to more closely examine and describe these differences, all subsequent analyses will involve descriptions of single subjects rather than group data.

Insert Table 1 about here

Analysis of Performance of Individual Subjects on Simple Problems

The goal of the analysis in this section is to provide a formal description of the knowledge that each subject has concerning how the calculator solves simple problems. Thus, for each subject, a model was developed which could generate the subject's answers on simple test problems. This section describes the data source, the format of the models, how the data for a subject were fit by a particular model.

Data Source. This analysis was based on the data for the 33 novices and the 33 experts who gave consistent answers (i.e., high reliability between the two corresponding question sets). In order to provide an intensive analysis of the performance of each subject in the sample, this analysis focused on only the simple problems. Simple problems are defined as those which contain symbols for number and/or plus and/or equals but which do not contain any multiplication symbols. Eighteen of the 44 problems in set A fit this description; these are problems 1 through 10 and 27 through 34 in Table 1. Thus, for each of the 33 novices and 33 experts, the data source was a list of 18 answers for the 18 simple problems.

Development of Models. The goal of the present analysis was to develop simple production system models that would generate the performance of each subject on the 18 simple problems. Because of its efficiency and apparent relevance for the present task a production system was used to represent the knowledge of each subject. A production system contains a list of productions with each production consisting of a condition and a corresponding action. Conditions, actions, and productions for the current problem are described in this section.

The relevant conditions for the present analysis are related to having pressed one of the keys on the calculator keyboard. The key relevant to the simple problems are the ten number keys (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), the addition operation key (+) and the equals key (=). Table 2 shows nine conditions that could exist; three are not relevant to the simple problems since these conditions are never found in the problems. The other six conditions contain an exhaustive list of the conditions present in the simple problems. At first blush, it might seem that Table 2 gives many redundant conditions and that

a simplier scheme is to deal with only three basic conditions--pressing a number (#), pressing a plus (+), or pressing an equals (=). However, the performance and comments of subjects suggest that a key press means something different to some people depending on the immediately preceeding key press.

Insert Table 2 about here

The actions for each production consist of events that take place inside the calculator. Following a system developed to represent the "internal" actions in computer languages (Mayer, 1979), each of the calculator actions can be represented as a triplet: some operation is applied to some object at some location in the calculator. The operations consist of the following:

(1) Create -- A number or expression is placed in the display or register, e.g., when you press a number key that number appears in the display.

(2) Destroy -- A number or expression is erased from the display or register, e.g., when you press the equals key the previous number in the display is erased (and replaced with a new one).

(3) Evaluation -- An expression (from the register) is converted into a single number, e.g., the evaluation of 3 + 2 is 5. (For the current example, evaluation of a number or a number followed by an operation is the number, e.g., evaluation of 3 is 3, evaluation of 2 + is 2).

The objects consist of:

(1) Numbers -- A number is any single or multiple digit sequence such as 2, 14, 156, etc.

(2) Operation -- An operation is a mathematical symbol for some arithmetic computation such as addition (+), or multiplication (x).

(3) Expression -- An expression is a sequence consisting of numbers and operations such as, 2 + 3 or 2 + or 2 + 3 x 4.

The locations consist of:

(1) Display -- The external display in a calculator normally consists of at least eight places, where each place can hold one digit.

(2) Register -- An internal register is inside the calculator and consists of subregisters for individual numbers and for operations. Expressions are held in the order of input, with the first number on the left, followed by the operation, followed by the next number.

Table 3 shows some typical actions that may occur for the simple problems. It should be noted that the 12 actions listed in the table actually refer to groups of single actions. For example, D = # consists of two single actions: erasing the old number from the display and replacing it with a new number. Also, it should be noted that the first five actions refer only to the display, and have no effect on changing the register; the other actions refer only to the register and have no effect on changing the display.

Insert Table 3 about here

Since Table 2 gives a list of all relevant conditions and Table 3 gives a list of relevant actions, it is now possible to describe any production as one of the conditions coupled with one of the actions. For example, the production,

P5. If # after + Then D = # and R = R + #

means that when the number key is pressed after the plus key the result is that the old number in the display is replaced with the new number, and the old expression in the register is retained but the new number is added to the right. Thus, if the sequence had been 5 + 2 + and now a 3 is pressed, the display will contain a 3 (before there was a 2) and the register will contain the expression 5 + 2 + 3 (before there was 5 + 2 +).

Fitting a model to each subject. The foregoing sections described that the data source for each subject was the 18 answers to the target questions and that models are based on (alternative) actions associated with each of the six conditions. Thus, for the data of each subject the goal is to develop a production system which consists of six productions.

The task was made somewhat easier by the fact that there are groups of subjects who gave identical answers for the simple problems. Based on their answers to the 18 simple problems, subjects could be grouped in one of six distinct categories where subjects in each of the first five groups produced identical answers with one another. Group 1 contained 8 novices and 11 experts, all of whom gave answers that were identical to answers produced by inexpensive TI models. Group 2 contained 10 novices and 7 experts, all of whom gave answers that were identical to Group 1 except for situations in which a plus key was pressed. Group 3 contained 5 novices and 2 experts, all of whom gave answers that were identical to Group 1 except for situations in which a number key was pressed. Group 4 contained 2 novices and 4 experts, all of whom gave answers

that were similar to those produced by an inexpensive Rockwell model. Group 5 contained one novice and 3 experts, all of whom showed a mixture of acting like Group 1 and Group 4. Finally, there were 7 novices and 6 experts who gave idiosyncratic patterns of answers for the simple problems. Some of the subjects in this category were very similar to one of the above groups except for one or two minor deviations, while others seemed to have highly unique answers. In all cases, however, the answers were internally consistent within subjects as indicated by high correspondence between answers in set A and set B. The answers for each of the common categories on each of the 18 simple problems are given in Table 4.

Insert Table 4 about here

Models were generated for each of the common categories of answers and for each subject in the miscellaneous category. The production systems for each of the five common categories are given in Tables 5 through 9, respectively.

Insert Tables 5 through 9 about here

Groups 1, 2, and 3 all behave similarly but differ with respect to when an expression is evaluated and displayed. For example, consider the sequence of key strokes: 2 + 3 + 4 = .

According to model 1, subjects think that the calculator evaluates expressions only when a plus key (+) is pressed after a number or when an equals key (=) is pressed. Thus, in the above example, the number in the display after each of the six key strokes will be, 2, 2, 3, 5, 4, 9. The 3 does not get added to the 2 until a plus (or an equals) key is pressed; and the subtotal 5 does not get added to 4 until an equals (or plus) key is pressed.

According to model 2, subjects think that the calculator evaluates expressions only when an equals (=) key is pressed. Thus, in the above example, the numbers in the display after each of the six key strokes will be 2, 2, 3, 3, 4, 9. The entire expression is held in the register until an equals key is pressed.

According to model 3, subjects think that the calculator evaluates expressions as soon as a number key (#) is pressed. Thus, in the above example, the numbers in the display after each respective key stroke will be: 2, 2, 5, 5, 9, 9. The 3 gets added to the 2 as soon as the 3 is pressed; the 4 gets added to the subtotal 5 as soon as the 4 is pressed.

In summary model 2 involves delayed evaluation (i.e., nothing gets evaluated until an equals is pressed), model 3 involves immediate evaluation (i.e., expressions are evaluated as soon as possible), and model 1 involves a compromise between the two extremes (i.e., expressions are evaluated after a plus but not after a number key is pressed). Another way to describe the differences among the first three groups is to say that group 1 treats a plus key like an equals, group 3 treats both a number key and a plus key like an equals, group 2 treats neither like an equals.

The fourth and fifth groups differ from the first three groups with respect to how to deal with the plus key. The fourth group behaves as if the calculator has an automatic constant--evaluation of an expression for a plus or an equals involves adding the number in the display to the number in the register. This mode of evaluation is called "incrementing display" in the tables. The fifth group gives an "incrementing display" only when two plus keys are pressed in sequence. Both groups are like group 1 in that they display the evaluated version of the expression when a plus key or an equals key is pressed but not

when a number key is pressed. The "incrementing display" characteristic of group 4, and to some extent group 5, is a more sophisticated and efficient feature of some calculators such as Rockwell.

Insert Table 10 about here

Table 10 gives a list of alternative actions associated with each of the first 6 productions. As can be seen, the five common categories resulted in three alternatives for production P2 (P2A, P2B, P2C), three for P4 (P4A, P4B, P4C), two for P5 (P5A, P5B), one for P6 (P6A), two for P7 (P7A, P7B), and three for P8 (P8A, P8B, P8C). In the process of developing models for the 14 miscellaneous subjects, several new alternatives were constructed as shown in Table 10. Although a detailed analysis of the performance of each miscellaneous subject would require undue space, examples are given in this section. For example, one of the novices gives the answers 2, 2, 5, 5, 12, 2, 0, 5, 5, 12, 0, 0, 0, 3, 0, 0, 0, 3 for the 18 problems listed in Table 4, respectively. This subject seems to evaluate expressions immediately when a number key is pressed, corresponding to subjects in group 3. However, in addition, this subject treats any irregular sequence of button presses (such as + after +, or + after =) as a reset or clearing of the display and register. The productions which describe this subject's performance are: P2B, P4B, P5C, P6D, P7B and P8B. As another example, one of the experts gives the answers, 2, 2, 3, 5, 7, 2, 4, 5, 10, 12, 2, 4, 8, 3, 4, 8, 16, 7, for the 18 simple problems, respectively. This performance is similar to model 1 except that the display is incremented for = after +. The productions are P2A, P4A, P5A, P6A, P7A, P8C. Models were fit to each of the miscellaneous subjects by taking the best fitting common model (i.e., models 1 through 5) and changing as few productions as necessary in order to make the fit perfect.

The left side of the Table 10 lists the frequencies of each production for the experts and novices. There is a tendency for experts to rely on productions which involve incrementing the display, i.e., P5B and P8C. For example, these productions are used 6 times by novices and 14 times by experts. In addition, there is a tendency for experts to rely on productions which evaluate and display for + and = but not for #, i.e., P2A, P4A, P7A, while novices tend to favor immediate evaluation and display for #, i.e., P2B, P4B, P7B. For example, the former set of productions is used 76 times by experts and 58 times by novices out the latter set is used 13 times by experts and 31 times by novices.

Analysis of Individual Subjects on Multiplication Problems

The previous section encouraged the idea that it is possible to describe the subject's "model of the calculator" for generating answers to 18 simple problems. These analyses were based on problems containing only six possible conditions. In the present section, we expand our analysis to include 16 additional problems which contain multiplication. These are problems 11 through 26 on Table 1. They provide three new conditions: x after # (i.e., pressing the multiply key after pressing a number key, such as 2 x), # after x (i.e., pressing a number key after pressing a multiply key, such as 2 x 3) and = after x (i.e., pressing an equal key after pressing a multiply key, such as 2 x =). Also, up to this point we have considered only conditions which include two events, but this group of questions also allows us to explore whether subjects use more than two events to determine chains of arithmetic; for example, if a subject evaluated all multiplications before additions, then 2 + 3 x 7 = would yield an answer of 23 but if a subject evaluated chains in order of presentation then the answer is 35. Thus, this analysis will allow us to add three new productions to each subject's model developed in the previous section, and to modify some productions for evaluating chain arithmetic.

For each subject, we assume that the six productions established in the previous analysis still are operating for problems 11 through 26. Thus, the goal of the present section is simply to add three new productions (P10, P11, P12) to the model of each subject. All 16 problems contain the condition x after #; almost all of the problems contain the condition, = after x; problems 15, 17, and 23 provide the condition = after x; and several problems involve chain of + and x operations such as problems 24 and 26.

Table 10 shows the most common possible actions associated with each of the three new conditions (P10, P11, P12) explored in this section. As with the analysis of simple problems, one issue concerns when an expression is evaluated. If an expression is evaluated as soon as a multiplication operation (x) or an equals (=) key is pressed, analogous to Model 1 in the previous analysis, then the productions selected would be: P10B (no evaluate for #), P11A (evaluate for x), and P12A (evaluate for equals). If an expression is evaluated only when an equals key is pressed, analogous to the delayed evaluation in Model 2, then the productions selected would be: P10B (no evaluate for #), P11B (no evaluate for x), P12A (evaluate for =). If an expression is evaluated as soon as a number key is pressed, analogous to the immediate evaluation of Model 3, then the productions selected would be: P10A (evaluate for #), P11B (no evaluate for x), P12B (no evaluate for =). Finally, if subjects used an incrementing display for evaluating expressions and numbers as in Model 4 or 5 in the previous section, then the selected productions would be: P10B (no evaluate for #), P11B (no evaluate for x), and P12C (increment display for =). For purposes of this analysis we will refer to each of these four clusters of three productions as Model 1m, Model 2m, Model 3m, and Model 4m, respectively.

Table 11 shows the answers generated by each of the four multiplication models (i.e., productions P10, P11, and P12). Thirty of the 33 novices and 23 of the 33 experts generated performance on the multiplication problems that was consistent with one of the four models; however, specific answers to some problems could differ from those listed in Table 11 in cases where different systems for evaluating chains of arithmetic or different productions for P2 through P8 were in use. The bottom of Table 11 shows the number of novices and experts who were fit by each of the four models.

Insert Table 11 about here

Model 2m allows for evaluation of a chain of arithmetic. Of the novices, 5 subjects performed the arithmetic in order from left to right (as indicated in Table 11), one subject carried out additions before multiplications (e.g., 2 x 3 + 7 = resulted in an answer of 20), one subject carried out multiplications before additions (e.g., 2 + 3 x 7 = resulted in an answer of 23) and one carried out computations only on the last two entries in the register (e.g., 2 x 3 x 7 = resulted in 21; or 2 x 3 + 7 = resulted in 10). Of the experts, three out of five subjects showing model 2m performed chains from left to right, and two of the five experts performed multiplication before addition in a chain.

There were also three miscellaneous novices and 10 miscellaneous experts. Of the novices, two subjects gave model 1m answers for problems that involved only numbers, multiplication, and/or equals but model 2m answers for problems with numbers, addition, multiplication, and equals. One novice gave model 4m answers for problems with numbers, multiplication and/or equals but model 3m answers when problems involved multiplication, addition, numbers and equals.

Thus, these subjects behaved as if the conditions for actions depended on more than just the last two button presses. No additional productions were constructed to try to fit this performance in Table 11. However, for the 10 unique experts, several additional productions for P10, P11 and P12 were constructed and are listed in Table 10. For example, one subject reset the display to zero for x after # and to no change for = after x. The productions for that subject are P10B, P11C, P12A. Another subject ignored the equal sign when it followed the multiplication sign, as indicated by productions P10B, P11B, P12B. Two other subjects reset the display for = after x giving the productions P10B, P11B, P12C. Thus, many of the experts tend to add new productions for unusual button sequences; the effect of most of the new productions is some sort of "reseting" the display. The frequency of use of each alternative production for P10, P11, and P12 for all subjects is summarized in the left side of Table 10.

Analysis of Individual Subjects on Complex Problems

Finally, the performance of each subject on problems 35 through 44 was analyzed. These problems contain many of the conditions already described in the previous two sections; thus, it was assumed that each subject would use the twelve productions already determined by analyzing the first 34 problems in the test. However, problems 35 through 44 also contain four new conditions: x after =, x after x, + after x, and x after +. Thus, in this section four new productions (P13, P14, P15 and P16) are added to the model of each subject.

Table 10 lists the alternative productions for each of the four new conditions explored in this section. Table 12 gives some typical answers by subjects for problems 35 through 44.

Insert Table 12 about here

Group 1 in Table 12 behave as if they were using productions 13A, 14A, 15A, and 16A along with delayed evaluation of expressions (based on earlier productions). These subjects treat x after = and x after x as if there is no change, and if two consecutive operation keys are pressed (such as x after +) they use only the last operation that was pressed. As shown in the bottom of Table 12, there were ten novices and 15 experts who followed this procedure. There were also six more novices and one more expert in group 1'; these subjects behave as if they use the identical four new productions but they show immediate evaluation of expressions when the number key is pressed. In addition, four novices (and no experts) in group 2 behave like those in group 1 except that when there are two consecutive operations, the multiply "wins", i.e., for 2x+ the register stores 2x. The productions for this group are P13A, P14A, P15B, P16A. Similarly, two more novices and one expert in group 2' use the same four new productions as group 2 but act as if an expression is evaluated as soon as a number key is pressed. In addition, there were 2 novices and 4 experts in group 3. These subjects treat the four new productions as if they serve to increment the display. This procedure is indicated by the combination of P13A, P14B, P15C, P16C. These are the same subjects who increment the display for similar conditions such as + after + or + after = or = after +.

There were also a large number of unique subjects--nine novices and 12 experts. However, almost all of the subjects are closely related to either model 1 or model 3, with just one production slightly different. For example, one subject is like model 1 except that the display is reset to zero for x after + or = after x. The productions for that subject are P13A, P14A, P15D, P16D. Another subject has the same procedure as subjects in group 2 except that x after x results in the display being multiplied by the register; the productions

for this subject are P13A, P14B, P15B, P16A. For a third subject, the display is incremented for = after x and x after = and when two consecutive operations are input the multiply wins; the productions are, P13C, P14B, P15B, P16A. The left portion of Table 10 summarizes the frequencies of each of the alternative productions for all subjects on P13, P14, P15 and P16.

General Conclusions

Characterizing the Differences Among Subjects

The present study suggests that subjects differ with respect to their conceptions of the operation of electronic calculators. The foregoing analyses (summarized in Table 10) provided for a detailed description of the differences among subjects, with each subject being described as a list of productions. However, the goal of this section is to provide a more integrated description of the major differences among subjects. Three basic kinds of differences were observed: (1) How is an expression represented in the register? For example, a series of key strokes such as 2+x3 can be represented as 2 + 3 or 2 x 3 or 0 or something else. (2) When is an expression evaluated? For example, does the calculator evaluate at the earliest possible opportunity such as 2 + 3 resulting in a display of 5, does the calculator wait for an equals to be pressed before an evaluation takes place, or does the calculator compromise between these two extremes? (3) How is an expression evaluated? For example, a chain of arithmetic like 2 + 3 x 7 can be evaluated from left to right (answer is 35) or with multiplication first (answer is 23) or in some other way; furthermore, non-standard sequences such as 2+= can be evaluated by incrementing the display to 4 or by ignoring the plus (display is 2) or by reseting the display to 0. In this section, we explore how the subjects differ with respect to their conceptions of how to represent expressions, when to evaluate, and how to evaluate.

Standard conditions. First, there are some general differences which emerge by investigating differences for standard conditions such as # after +, # after x, + after #, x after #, = after #. These are sequences that follow the standard grammar of arithmetic, and are listed as P2, P10, P4, P11, and P7 in Table 10.

The first issue of how to represent expressions is fairly straightforward for all subjects--symbols are added to the register in order from left to right. For example, the keystrokes 2 + 3 x 7 is represented in exactly that way in the register.

The second issue is when to evaluate the expression. In our analysis we located three distinct approaches to the question of when to evaluate. The compromise method is to evaluate an expression whenever an equals key or an arithmetic operation key is pressed but not when a number key is pressed; the immediate method is to evaluate as soon as a number key is pressed (e.g., for 3 + 5 display shows 8); the delayed method is to evaluate only when an equals key is pressed (e.g., for 3 + 5 the display shows 5). The novices and experts tend to differ with respect to their consensus on when to evaluate. Of the novices 13 tend to opt for compromise evaluation, 13 for delayed evaluation, and 7 for immediate evaluation; for experts there is a much stronger consensus of 24 subjects favoring compromise evaluation with 7 favoring delayed evaluation and 2 favoring immediate. A chi-square test revealed that novices and experts differ significantly with respect to the proportion of subjects favoring compromise evaluation, $\chi^2 = 6.15$; df = 1, $p < .05$.

The third issue concerns how to evaluate an expression. In most cases, subjects overwhelmingly follow the normal rules of arithmetic. However, as noted earlier, when subjects use delayed evaluation they may be confronted with a chain of arithmetic such as 2 + 3 x 7. While the majority of subjects eval-

uate a chain from left to right (i.e., generating an answer of 35), some experts carry out multiplication before addition (i.e., answer is 23), and some novices use other schemes such as carry out additions before multiplications or carry out only the last computation (i.e., answer is 21).

Non-standard conditions (equals after operation). In the present study we also investigated subjects' interpretations of several non-standard conditions, such as = after + or = after x. These are conditions which violate the grammar that demands a number between the operation symbol and the equals symbol. Table 10 represented these as P8 and P12.

The main issue here is how to represent and evaluate an expression when the last key press was an operation and now an equals is pressed. For production P8, the majority of novices (n = 27) and the majority of experts (n = 25) ignore the last + key that was pressed. Thus, a sequence like 2+= results in a display of 2, or a sequence like 2+3+= results in a display of 5. We call this the "no effect" approach because subjects act as if the plus key had no effect. A second approach is what we call the "incrementing" approach; here subjects create some number to go between the + and the = such as the number in the display. For example, the sequence 2+= results in a display of 4 (i.e., it is treated as 2+2=), or 2+3+= may result in 10 (i.e., it is treated as 5+5=) or 8 (i.e., it is treated as 2+3+3=). There were three novices and 6 experts who opted for the incrementing approach. A third option is what we call the "reset" approach. Here subjects reset the display to some number (such as zero) for any non-standard sequence of key presses. Three novices and two experts used a version of the reset approach. The comparable figures for production P12 were 29 novices and 25 experts favored the no effect approach while 4 novices and 8 experts favored the incrementing approach. Although the proportion of subjects

favoring the incrementing approach is twice as high for experts as for novices, chi-square tests failed to indicate that the proportion of incrementing subjects was greater for experts in P8, $\chi^2 = .61$, df = 1, or P12, $\chi^2 = .92$, df = 1.

Non-standard conditions (two consecutive operations). Another type of non-standard condition investigated in this study was two consecutive operations such as + after +, x after x, + after x, or x after +. These are conditions which violate the grammatical demand for a number between any two operation symbols. Table 10 presents these as P5, P14, P15, P16.

The main issue here is how to represent and/or evaluate an expression when the last key presses are two arithmetic operators. The three major options chosen by our subjects correspond to those discussed above: "no effect" involves selecting one of the two operation signs to be included in the register and ignoring the other; for example, the most common version of this approach is to ignore the second operation so that 2++ is represented in the register as 2+ or 2xx is represented as 2x. "Incrementing" involves selecting a number to be inserted between the operator symbols; the most common version of this approach is to insert the number from the display so that 2++ becomes 2+2+ or 2xx becomes 2x2x. "Reset" involves clearing the display such as setting it to zero; for example, 2++ results in 0 being displayed. For production P5 the majority of novices (n = 28) and experts (n = 24) opted for the no effect approach; in addition four novices and nine experts opted for the incrementing approach; and one novice and no experts reset the display. The figures for production P14 are similar: 28 novices and 24 experts opted for no effect; incrementing was opted for by 4 novices and 8 experts; and one novice and one expert opted for the reset approach. The patterns for P15 and P16 are similar--the majority of each novices and experts opt for no effect but a substantial

number of experts opt for the incrementing option. In all productions, the proportion of experts who increment is more than twice that of the novices. However, even in the most extreme case, the differences in proportion of "incrementers" between experts and novices fails to reach statistical significance, $\chi^2 = 2.27$, df = 1.

Non-standard conditions (operation after equals). Finally, our test involved two productions P6 and P13, which involve + after = and x after = respectively. The status of the register after an = key is pressed is that it contains a number. Thus, these non-standard conditions are most frequently treated in the same way that + after # or x after # is treated. For P6, 30 of the novices and 32 of the experts use this "no effect" approach of simply adding a plus sign to the register. For P13, 28 novices and 27 experts follow the "no effect" approach of adding a multiply sign to the register. However, a sizable minority of the experts (n = 6) opt for an incrementing approach while a sizable minority of the novices opt for a reset option (n = 4).

Summary. The present study provides new information concerning how humans think about calculators. First, we were able to apply the analytic techniques of cognitive psychology to a real-world domain. This allowed a formal and detailed description of how each subject interpreted what was going on inside the "black box" when a key was pressed. Second, in spite of the fact that all of our subjects were proficient in using a calculator to solve standard computational problems, we observed tremendous individual differences among users in their interpretations of the logic of the calculator's operating system. Thus, in spite of apparent similar performance on standard problems, people differ greatly in their knowledge of how the calculator solves problems.

Experts tended to give more consistent answers, as would be expected; however, they also tended to prefer certain operating characteristics such as evaluating an expression for either an operation key or an equal key (compromise evaluator), and incrementing the display during evaluation with non-standard conditions. Further work is needed to determine whether people with certain sets of intuitions can use their calculators more creatively or can learn a new computer language (such as programmable calculators or BASIC) more efficiently than people with other sets of intuitions. In addition, future work is needed to determine whether intuitions--once they have been diagnosed--can be altered through instruction. It is hoped that the groundwork laid in this study will serve as an incentive for continued work in the development of a theory of computer literacy.

Reference Note

1. Suydam, M. N. State of the art review on calculators: Their use in education. Columbus, Ohio: Calculator Information Center, Report No. 3, 1978.

References

Bell, M. Calculators in elementary schools? Some tentative guidelines and questions based on classroom experience. Arithmetic Teacher, 1976, 23, 502-509.

Brown, J. S. & Burton, R. R. Diagnostic models for procedural bugs in basic mathematical skills. Cognitive Science, 1978, 3, 155-192.

Gaslin, W. L. A comparison of achievement and attitudes of students using conventional or calculator-based algorithms for operation positive rational numbers in ninth-grade general mathematics. Journal for Research in Mathematics Education, 1975, 6, 95-108.

Gawronski, J. D. & Coblentz, D. Calculators and the mathematics curriculum. Arithmetic Teacher, 1976, 23, 510-12.

Groen, G. J. & Parkman, J. M. A chronometric analysis of simple addition. Psychological Review, 1972, 79, 329-343.

Hopkins, E. E. A modest proposal concerning the use of hand held calculators in schools. Arithmetic Teacher, 1976, 23, 657-9.

Larkin, J. Information processing models and science instruction. In J. Lochhead and J. Clement (Eds.), Cognitive process instruction. Philadelphia, Franklin Institute Press, 1979.

Mayer, R. E. A psychology of learning BASIC. Communications of the ACM, 1979, 22, 589-593.

Meyer, P. I. When you use a calculator you have to think½ Arithmetic Teacher, 1980, 27, 19-21.

Mullish, H. The complete pocket calculator handbook. New York: Collier, 1976.

National Council of Teachers of Mathematics, Instructional Affairs Committee. Minicalculators in schools. Arithmetic Teacher, 1976, 23, 72-6.

Newell, A. & Simon, H. A. Human problem solving. Englewood Cliffs, N. J.: Prentice-Hall, 1972.

Roberts, D. M. The impact of electronic calculators on educational performance. Review of Educational Research, 1980, 50. 71-98.

Roberts, D. M. & Fabrey, L. J. Effects of calculator method, amount of pre-practice and instructional work set on numerical problem solving. Calculators/Computer Magazine, 1978, 2, 63-70.

Roberts, D. M. & Glynn, S. M. Teaching introductory statistics for psychology: Effects of calculators on performance and motivation. Perceptual and Motor Skills, 1979, 48, 563-569.

Rudnick, J. A. & Krulik, S. The minicalculator: Friend or foe? Arithmetic Teacher, 1976, 23, 654-6.

Scandura, A. M., Lowerre, G. F., Veneski, J. & Scandura, J. M. Using electronic calculators with elementary school children. Educational Technology, 1976, 16, No. 8, 14-18.

Shneiderman, B. Software psychology: Human factors in computer and information systems. Cambridge, Mass.: Winthrop, 1980.

Schnur, J. O. & Lang, J. W. Just pushing buttons or learning? A case for minicalculators. Arithmetic Teacher, 1976, 23, 559-63.

Simon, D. P. & Simon, H. A. Individual differences in solving physics problems. In R. Siegler (Ed.), Children's thinking: What develops. Hillsdale, N. J.: Erlbaum, 1978.

Woods, S. S., Resnick, L. B. & Groen, G. J. An experimental test of five process models for subtraction. Journal of Educational Psychology, 1975, 67, 17-21.

Footnotes

This research was supported by grant NIE-G-78-0162 from the National Institute of Education. This project was initiated while the senior author was on sabbatical leave at the Learning Research and Development Center, University of Pittsburgh. We appreciate the comments of Richard Young on an earlier version of this project. Requests for reprints should be sent to Richard E. Mayer or Piraye Bayman, Department of Psychology, University of California, Santa Barbara, CA 93106.

[1]Twelve subjects were from the University of Pittsburgh and 34 subjects were from the University of California, Santa Barbara. However, since there were no systematic differences between these groups in age, sex, GPA, SAT scores, or answers to the experimental test, they have been combined into one larger sample.

[2]We wish to thank Dr. Larry Lichten of the Computer Science Department of the University of California, Santa Barbara, for his help in locating subjects for study 2.

[3]For each problem in which the subject gave inconsistent answers between set A and set B, the answer to the A set was used unless it was inconsistent with related items.

[4]Since the HP-21 uses reverse Polish notation (RPN) it was not used as a model in this study.

Table 1

Frequencies of Answers for 44 Problems

Problem Number	Problem	Study 1 Modal Answer	Study 1 Alternatives	Study 2 Modal Answer	Study 2 Alternatives
1	2	(2)=33		(2)=33	
2	2+	(2)=33		(2)=33	(0)=1
3	2+3	(3)=25	(5)=8	(3)=32	(5)=1
4	2+3+	(5)=21	(3)=12	(5)=23	(3)=9;(0)=1
5	2+3+7	(7)=26	(12)=6;(15)=1	(7)=32	(12)=1
6	2=	(2)=33		(2)=33	
7	2+=	(2)=29	(4)=3;(0)=1	(2)=27	(4)=6
8	2+3=	(5)=33		(5)=33	
9	2+3+=	(5)=30	(10)=3	(5)=26	(10)=5;(3)=1;(8)=1
10	2+3+7=	(12)=29	(10)=2;(11)=1;(15)=1		
(12)=33					
11	2x	(2)=33		(2)=32	(0)=1
12	2x3	(3)=24	(6)=9	(3)=33	
13	2x3x	(6)=24	(3)=9	(6)=23	(3)=9;(0)=1
14	2x3x7	(7)=24	(42)=9	(7)=32	(6)=1
15	2x=	(2)=29	(4)=4	(12)=25	(4)=7;(0)=1
16	2x3=	(6)=33		(6)=33	

Table 1 (Con't.)

Frequencies of Answers for 44 Problems

Problem Number	Problem	Study 1 Modal Answer	Study 1 Alternatives	Study 2 Modal Answer	Study 2 Alternatives
17	2x3x=	(5)=29	(36)=4	(6)=23	(36)=6;(3)=2;(18)=1;(0)=1
18	2x3x7=	(42)=31	(21)=1;(7)=1	(42)=32	(41)=1
19	2+3x	(5)=22	(3)=11	(5)=18	(3)=13;(0)=1;(6)=1
20	2+3x7	(5)=25	(35)=7;(42)=1	(7)=32	(42)=1
21	2x3+	(6)=24	(3)=9	(6)=25	(3)=7;(0)=1
22	2x3+7	(7)=26	(13)=7	(7)=32	(13)=1
23	2+3x=	(5)=26	(25)=3;(3)=2;(11)=1;(2)=1	(5)=23	(25)=3;(3)=3;(11)=2;(2)=1;(15)=1
24	2+3x7=	(35)=29	(23)=2;(21)=1;(42)=1	(35)=23	(23)=7;(42)=2;(45)=1
25	2x3+=	(6)=29	(12)=3;(3)=1	(6)=26	(12)=5;(3)=1;(18)=1
26	2x3+7=	(13)=29	(42)=2;(10)=1;(20)=1	(13)=31	(11)=1;(42)=1
27	2++	(2)=29	(4)=3;(0)=1	(2)=23	(4)=9;(0)=1
28	2+=+	(2)=27	(4)=4;(0)=2	(2)=25	(4)=8;(0)=1
29	2+=+=+	(2)=26	(8)=3;(0)=2;(4)=1;(6)=1	(2)=23	(8)=5;(4)=2;(6)=2;(0)=1
30	2+=+3	(3)=28	(5)5	(3)=32	(2)=1
31	2++=	(2)=27	(40=3;(8)=2;(0)=1	(2)=22	(4)=6;(8)=4;(6)=1
32	2+=+=	(2)=28	(8)=2;(4)=1;(6)=1;(0)=1		
(2)=24		(8)=5;(4)=3;(6)=1			

Table 1 (Con't.)

Frequencies of Answers for 44 Problems

Problem Number	Problem	Study 1 Modal Answer	Study 1 Alternatives	Study 2 Modal Answer	Study 2 Alternatives
33	2+=+=+=	(2)=28	(16)=3;(8)=1;(6)=1	(2)=24	(16)=5;(6)=3;(8)=1
34	2+=+3=	(5)=26	(7)=4;(3)=3	(4)=26	(7)=6;(3)=1
35	2xx	(2)=29	(4)=3;(0)=1	(2)=23	(4)=9;(0)=1
36	2x=x	(2)=28	(4)=4;(0)=1	(2)=24	(4)=8;(0)=1
37	2x=x=x	(2)=27	(16)=3;(4)=1;(8)=1;(0)=1	(2)=21	(16)=5;(8)=4;(4)=2;(0)=1
38	2x=x3	(3)=24	(6)=9	(3)=29	(6)=3;(12)=1
39	2xx=	(2)=26	(4)=4;(8)=1;(0)=1;(16)=1	(2)=20	(4)=7;(16)=3;(8)=2;(0)=1
40	2x=x=	(2)=27	(16)=3;(0)=1;(4)=1;(8)=1	(2)=22	(16)=5;(8)=3;(4)=2;(0)=1
41	2x=x=x=	(2)=27	(256)=3(8)=1;(16)=1;(0)=1	(2)=21	(256)=5;(16)=2;(8)=3;(32)=1;(0)=1
42	2x=x3=	(6)=24	(3)=5;(12)=3;(48)=1	(6)=24	(12)=7;(3)=1;(0)=1
43	2x+3=	(5)=16	(6)=13;(7)=2;(8)=1;(3)=1	(5)=21	(7)=6;(6)=3;(e)=2;(3)=1
44	2+x3=	(6)=25	(5)=4;(3)=2;(12)=2	(6)=24	(12)=6;(5)=1;(7)=1;(e)=1

Note. - Number in parentheses indicates answer; number to right of equals indicates frequency.

Table 2

Conditions for Simple Problems

Name	Condition	Example		Description
P1	# after #	2	3	Pressing a number key after pressing a number key.
P2*	# after +	+	3	Pressing a number key after pressing a plus key.
P3	# after =	=	3	Pressing a number key after pressing an equals key.
P4*	+ after #	2	+	Pressing a plus key after pressing a number key.
P5*	+ after +	+	+	Pressing a plus key after pressing a plus key.
P6*	+ after =	=	+	Pressing a plus key after pressing an equals key.
P7*	= after #	3	=	Pressing an equals key after pressing a number key.
P8*	= after +	+	=	Pressing an equals key after pressing a plus key.
P9	= after =	=	=	Pressing an equals key after pressing an equals key.

Note.--Asterisk (*) indicates that production is relevant to simple problems. P1, P3 and P9 are not relevant since they do not occur in the simple problems.

Table 3

Some Action for Simple Problems

Action	Description
D = D	No change in the display
D = #	Erase the old number from the display. Put the new number in the display.
D = R	Erase the old number from the display. Copy the number from the register into the display.
D = eval(R)	Erase the old number from the display. Put the value for the expression in the register into the display.
D = eval(D+R)	Erase the old number from the display. Replace it with the value for the sum of that number from the display and the value in the register.
R = R	No change in register.
R = #	Retain the present expression in the register. Place a number to the right of the expression in the register.
R = "R+#"	Retain the present expression in the register. Place a number to the right of the expression in the register.
R = eval (R)	Erase the old number or expression from the register. Replace it with the evaluation of the number or expression.
R = eval (D+)	Erase the old number or expression from the register. Replace it with the sum of the number in the display plus the evaluation of the register.
R = eval (R)+	Erase the old expression or number from the register. Replace it with the evaluation of that number or expression, and follow that with a plus.

Table 4

Problems and Answers or 18 Simple Items for Four Groups of Subjects

Problem Number	Problem	Group 1 Answer	Group 2 Answer	Group 3 Answer	Group 4 Answer	Group 5 Answer	Miscellaneous
1	2	2	2	2	2	2	
2	2+	2	2	2	2	2	
3	2+3	3	3	5	3	3	
4	2+3+	5	3	5	5	5	
5	2+3+7	7	7	12	7	7	
6	2=	2	2	2	2	2	
7	2+=	2	2	2	4	2	
8	2+3=	5	5	5	5	5	
9	2+3+=	5	5	5	10	5	
10	2+3+7=	12	12	12	12	12	
27	2++	2	2	2	4	4	
28	2+=+	2	2	2	4	2	
29	2+=+=+	2	2	2	8	2	
30	2+=+3	3	3	5	3	3	
31	2++=	2	2	2	8	4	
32	2+=+=	2	2	2	8	2	
33	2+=+=+=	2	2	2	16	2	
34	2+=+3=	5	5	5	7	5	
Number of Subjects- Study 1		8	10	5	2	1	7
Number of Subjects- Study 2		11	7	2	4	3	6

Table 5

Production System for Model 1 (Compromise Evaluation)

Production Name	Condition	Action		Comments
P2A	If # after +	then Set D = #	Set R = "R + #"	
P4A	If + after #	then Set D = eval (R)	Set R = eval (R) +	
P5A	If + after +	then Set D = D	Set R + = R +	(NO CHANGE)
P6A	If + after =	then Set D = eval (R)	Set R = eval (R)	
P7A	If = after #	then Set D = eval (R)	Set R = eval (R)	
P8A	If = after +	then Set D = eval (R)	Set R = eval (R)	

Note.--Evaluates after + or =. Display is non-incrementing.

Table 6

Production System for Model 2 (Delayed Evaluation)

Production Name	Condition	Action		Comments
P2A	If # after +	then Set D = #	Set R = "R + #"	(SAME AS MODEL 1)
P4B	If + after #	then Set D = D	Set R = eval (R) +	
P5A	If + after +	then Set D = D	Set R + = R +	(SAME AS MODEL 1)
P6A	If + after =	then Set D = D	Set R = R +	(SAME AS MODEL 1)
P7A	If = after #	then Set D = eval (R)	Set R = eval (R)	(SAME AS MODEL 1)
P8A	If = after +	then Set D = eval (R)	Set R = eval (R)	(SAME AS MODEL 1)

Note.--Evaluates after =. Display is non-incrementing.

Table 7

Production System for Model 3 (Immediate Evaluation)

Production Name	Condition	Action		Comments
P2B	If # after +	then Set D = eval (R +#)	Set R = eval (R+#)	
P4B	If + after #	then Set D = D	Set R = eval (R) +	
P5A	If + after +	then Set D = D	Set R = R +	(SAME AS MODEL 1)
P6A	If + after =	then Set D = D	Set R = R +	(SAME AS MODEL 1)
P7B	If = after #	then Set D = D	Set R = R	(NO CHANGE)
P8B	If = after +	then Set D = D	Set R = R	(NO CHANGE)

Note.--Evaluate after #. Display is non-incrementing.

Table 8

Production System for Model 4 (Incrementing Display)

Production Number	Condition	Action		Comments
P2C	If # after +	then Set D = #	Set R = eval (R + #)	
P4C	If + after #	then Set D = R	Set R = R	
P5B	If + after +	then Set D = eval (D+R)	Set R = eval (D+R)	(INCREMENTING DISPLAY)
P6A	If + after =	then Set D = eval (R)	Set R = eval (R)	(SAME AS MODEL 1)
P7A	If = after #	then Set D = eval (R)	Set R = eval (R)	(SAME AS MODEL 1)
P8C	If = after +	then Set D = eval (D+R)	SEt R = eval (D+R)	(INCREMENTING DISPLAY)

Note.--Evaluate after + or =. Display is incrementing.

Table 9

Production System for Model 5 (Partially Incrementing Display)

Production Name	Condition	Action		Comments
P2A	If # after +	then Set D = #	Set R = "R + #"	(SAME AS MODEL 1)
P4A	If + after #	then Set D = eval (R)	Set R = eval (R) +	(SAME AS MODEL 1)
P5B	If + after +	then Set D = eval (D+R)	Set R = eval (D+R)	(SAME AS MODEL 4)
P6A	If + after =	then Set D = eval (R)	Set R = eval (R)	(SAME AS MODEL 1)
P7A	If = after #	then Set D = eval (R)	Set R = eval (R)	(SAME AS MODEL 1)
P8A	If = after +	then Set D = eval (R)	Set R = eval (R)	(SAME AS MODEL 1)

Note.--Compromise between model 1 and model 4.

Table 10

Frequencies of Productions for All Subjects

Frequency in Study 1	Frequency in Study 2	Production Number	Condition	Action	Description
21	27	P2A	If # after +	then Set D = #, Set R = "R+#"	Delayed evaluation and display
7	2	P2B	If# after +	then Set D = eval (R+#), Set R = eval (R+#)	Immediate evaluation and display
4	4	P2C	If # after +	then Set D = #, Set R = eval (R+#)	Immediate evaluation and delayed display
1	0	P2D	If # after +	then Set D = eval (R+D), Set R = eval (R+D)	Immediate evaluation and display with incrementing evaluation
11	18	P4A	If + after #	then Set D = eval (R), Set R = eval (R) +	Immediate evaluation and display
18	9	P4B	If + after #	then Set D = D, Set R = eval (R) +	Immediate evaluation and delayed display
4	4	P4C	If + after #	then Set D = R, Set R=R+	Delayed evaluation and display
0	2	P4D	If + after #	then Set D = 0, Set R = 0	Reset to zero
28	24	P5A	If + after +	then Set D = D, Set R+ = R+	No change

Table 10 (continued)

Frequency in Study 1	Frequency in Study 2	Production Number	Condition	Action	Description
4	9	P5B	If + after +	then Set D = eval (D+R), Set R = eval (D+R)	Immediate incrementing evaluation and display
1	0	P5C	If + after +	then Set D = 0, Set R = 0	Reset to 0
30	32	P6A	If + after =	then Set D = D, Set R = R+	No change in display, add + to expression in register
1	0	p6B	If + after =	then Set D = D, Set R = eval (D+R)	No change in display, immediate incrementing evaluation
1	0	P6C	If + after =	then Set D = 0, Set R = R+	Reset display to 0, add + to expression in register
1	0	P6D	If + after =	then Set D = 0, Set R = 0	Reset display and register to 0, evaluate sum of constant and value in display
0	1	P6E	If + after =	then Set D = D, Set R = eval (#+D)	No change in display

Table 10 (continued)

Frequency in Study 1	Frequency in Study 2	Production Number	Condition	Action	Description
26	31	P7A	If = after #	then Set D = eval (R), Set R = eval (R)	Immediate evaluation and display
6	2	P7B	If = after #	then Set D = D, Set R = R	No change
1	0	P7C	If = after #	then Set D = R, Set R = R	Display value in register
21	23	P8A	If = after +	then Set D = eval (R), Set R = eval (R)	Immediate evaluation and display
6	2	P8B	If = after +	then Set D = D, Set R = R	No change in display or register
2	5	P8C	If = after +	then Set D = eval (D+R), Set R = eval (D+R)	Immediate incrementing evaluation and display
2	0	P8D	If = after +	then Set D = eval (R), Set R = 0	Display the evaluation of the expression in the register, reset the register to 0
1	1	P8E	If = after +	then Set D = eval (D+#), Set R = R	Display the sum of the value in the display plus a constant, no change in register

Table 10 (continued)

Frequency in Study 1	Frequency in Study 2	Production Number	Condition	Action	Description
1	2	P8F	If = after +	then Set D = D, Set = 0	No change in display, set register to 0
8	6	P10A	If # after x	then Set D = eval (R*#), Set R = eval (R*#)	Immediate evaluation and display
25	27	P10B	If # after x	then Set D = #, Set R = R*#	Delayed evaluation and display
11	15	P11A	If x after #	then Set D = eval (R), Set R = eval (R)*	Immediate evaluation and display
19	16	P11B	If x after #	then Set D = D, Set R = eval (R)*	Immediate evaluation and no change in display
0	2	P11C	If x after #	then Set D = 0, Set R = R*	Delayed evaluation and reset display to 0
3	0		(other mixed)		
19	18	P12A	If = after x	then Set D = eval (R), Set R = eval (R)	Immediate evaluation and display

Table 10 (continued)

Frequency in Study 1	Frequency in Study 2	Production Number	Condition	Action	Description
7	7	P12B	If = after x	then Set D = D, Set R = R	No change
4	5	P12C	If = after x	then Set D = eval (D*R), Set R = eval (D*R)	Immediate incrementing evaluation and display
0	1	P12D	If = after x	then Set D = eval (D*R), Set R = eval (R)	Immediate incrementing display, immediate, evaluation for register
0	2	P12E	If = after x	then Set D = eval (R*D), Set R = eval (R)* D	Immediate evaluation and display in the constant increment
3	0	(other mixed)			
28	27	P13A	If x after =	then Set D = D, Set R = D*	Delayed evaluation and display
4	0	P13B	If x after =	then Set D = 0, Set R = 0	Reset to 0
1	3	P13C	If x after =	then Set D = eval (D*R), Set R = eval (D*R)	Immediate incrementing evaluation and display
0	1	P13D	If x after =	then Set D = eval (D*R), Set R = R	Immediate incrementing display, no change in register
0	2	P13E	If x after =	then Set D = R, Set eval (D*R)	Immediate incrementing evaluation, delayed display

Table 10 (continued)

Frequency in Study 1	Frequency in Study 2	Production Number	Condition	Action	Description
28	24	P14A	If x after x	then Set D = D, Set R = R*	Delayed evaluation and display
4	8	P14B	If x after x	then Set D = eval (D*R), Set R = eval (D*R)*	Immediate incrementing evaluation and display
1	1	P14C	If x after x	then Set D = 0, Set R = 0	Reset to 0
0	0	P14D	If x after x	then Set D = D, Set R = D*R	Immediate incrementing evaluation, no change in display
19	25	P15A	If + after x	then Set D = D, Set R = R+	Set register sign from * to +
10	2	P15B	If + after x	then Set D = D, Set R = R*	No change
2	4	P15C	If + after x	then Set D = eval (D*R), Set R = eval (D*R)	Immediate incrementing evaluation and display
2	1	P15D	If + after x	then Set D = 0, Set R = 0	Reset to 0
0	1	P15E	If + after x	then Set D = eval (D*R), Set R = eval (D*R)+	Immediate incrementing evaluation and display with register sign to +

Table 10 (Continued)

Frequency in Study 1	Frequency in Study 2	Production Number	Condition	Action	Description
28	25	P16A	If x after +	then Set D = D, Set R = R*	Set register sign from + to *
1	0	P16B	If x after +	then Set D = D, Set R = R+	No change
2	4	P16C	If x after +	then Set D = eval (D*R), Set R = eval (D*R)	Immediate incrementing evaluation and display
2	1	P16D	If x after +	then Set D = 0, Set R = 0	Reset to 0
0	3	P16E	If x after +	then Set D = eval (D*R), Set R = eval (D*R)*	Immediate incrementing evaluation and display with register sign set to *

Table 11

Problems and Answers for 17 Multipliction Items by Four Groups of Subjects

Problem Number	Problem	Group 1m Answer	Group 2m Answer	Group 3m Answer	Group 4m Answer	Miscellaneous
11	2 x	2	2	2	2	
12	2 x 3	3	3	6	3	
13	2 x 3 x	6	3	6	6	
14	2 x 3 x 7	7	7	42	7	
15	2 x =	2	2	2	4	
16	2 x 3 =	6	6	6	6	
17	2 x 3 x =	6	6	6	36	
18	2 x 3 x 7 =	42	42	42	42	
19	2 + 3 x	5	3	5	5	
20	2 + 3 x 7	7	7	35	7	
21	2 x 3 +	6	3	6	6	
22	2 x 3 + 7	7	7	13	7	
23	2 + 3 x =	5	5	5	25	
24	2 + 3 x 7 =	35	35	35	35	
25	2 x 3 + =	6	6	6	12	
26	2 x 3 + 7 =	13	13	13	13	
Number of Subjects-Study 1		11	8	7	5	3
Number of Subjects-Study 2		15	5	0	3	10

Table 12

Problems and Answers for 10 Complex Items by Five Groups of Subjects

Problem Number	Problem	Group 1 Answer	Group 1' Answer	Group 2 Answer	Group 2' Answer	Group 3 Answer	Miscellaneous
35	2 x x	2	2	2	2	4	
36	2 x = x	2	2	2	2	4	
37	2 x = x = x	2	2	2	2	16	
38	x = x = 3	3	6	3	6	3	
39	3 x x =	2	2	2	2	16	
40	2 x = x =	2	2	2	2	16	
41	2 x = x = x =	2	2	2	2	256	
42	2 x = x 3 =	6	6	6	6	12	
43	2 x + 3 =	5	5	6	6	7	
44	2 + x 3 =	6	6	6	6	12	
Number of Subjects-Study 1		10	6	4	2	2	9
Number of Subjects-Study 2		15	1	0	1	4	12

CHAPTER 4

TASK ANALYSIS, DEVELOPMENT OF EVALUATION INSTRUMENTS, AND DIAGNOSIS OF BUGS FOR THE BASIC LANGUAGE

Note

This article has been published as the following citation:

Mayer, R. E., & Bayman, P. Users' mental models of BASIC. Technical Report No. 82-2. Santa Barbara: University of California, 1982.

This article has been submitted for publication as follows:

Mayer, R. E., & Bayman, P. Users' misconceptions of BASIC programming statements following hands-on experience. Cognition & Instruction, under review.

This research was also presented at a professional meeting as the following:

Mayer, R. E., & Bayman, P. Beginning programmers' comprehension of BASIC statements. Psychonomic Society, Minneapolis, November 11-13, 1982.

Abstract

In the process of learning a computer language, beginning programmers may develop mental models for the language. A mental model refers to the user's conception of the "invisible" information processing that occurs inside the computer between input and output. In this study, thirty undergraduates learned BASIC through a self-paced, mastery manual and simultaneously had hands-on access to an Apple II computer. After instruction, the students were tested on their mental models for the execution of each of nine BASIC statements. The results show that beginning programmers--although able to perform adequately on mastery tests in program generation--possessed a wide range of misconceptions concerning the statements they had learned. For example, the majority of the beginning programmers had either incorrect conceptions for or no conceptions of statements such as INPUT A, READ A, and PRINT C. This paper presents a catalogue of beginning programmers' conceptions of "what goes on inside the computer" for each of nine BASIC statements.

Users' Misconceptions of BASIC Programming Statements Following Hands-On Experience

Learning BASIC

This paper provides new information concerning how beginning programmers learn BASIC. Suppose that you were going to teach a beginning programmer how to use a programming language such as BASIC. The typical instructional sequence involves an orderly presentation of the elementary statements. For each statement, the following information is generally presented: definition, grammar, format specifications, example programs, and printouts. The apparent goal of most instructional sequences is to teach the user to perform. The typical instructional sequence occasionally includes partial descriptions of internal processes or states in the computer--such as memory locations, input stacks, etc.--but such descriptions are relatively rare and unsystematic. Thus, relatively little attention seems to be paid to the instructional goal of teaching the user to understand.

What is Learned

The main focus of this paper concerns "what is learned" when a beginning programmer is taught a computer programming language such as BASIC, following a typical instruction sequence.

The outcome of learning--what is learned--can be viewed in two distinct ways:

Learning BASIC involves acquisition of new information. This idea asserts that learning a computer programming language is like learning any subject matter; the learner must acquire specific facts, skills, and rules. For example, the learner acquires new information such as format rules for when to use quotes in a PRINT statement or how to produce a conditional loop

using an IF statement. Acquisition of specific information is generally the explicit objective of instruction.

Learning BASIC involves acquisition of a mental model. This idea asserts that, during the course of learning, a user develops a conception of the invisible actions and states that occur in the computer between input and output. For example, the learner acquires new mental models such as the idea of memory spaces for holding numbers as in a counter set LET statement. Acquisition of mental models is generally not an explicit objective of instruction.

The distinction between the acquisition of new information and the acquisition of a mental model is just for practical purposes. If we conceive of the learner as an active, thinking individual, it becomes necessary to think of the two outcomes of learning as complements of each other. Most instructional effort is directed towards the acquisition of new information. However, understanding of how mental models are acquired in the process of learning the new information might be useful for the instructor as well as the designer. One hypothesis is that basic training in the essential information coupled with the opportunity to practice at a computer terminal will naturally lead to the acquisition of "useful" mental models--models that can enhance understanding of the computer language. The present study aims to explore this hypothesis.

Mental Models for BASIC

The present study explores the idea that learning of BASIC involves more than the acquisition of specific facts, rules, and skills. Beginning programmers also develop mental models for the language in the process of learning the essentials of BASIC. Users' models, however, may not be accurate or useful ones. That is, users develop individual conceptions of

"what is going on inside the computer" as well as learn specific facts.

Mayer (1979) has suggested a framework for describing the internal transformations that occur for elementary BASIC statements. In particular, any BASIC statement can be conceptualized as a list of transactions. A transaction is a simple statement asserting some action performed on some object at some location in the computer. For example, the following set of transactions describes what happens inside the computer when the statement 10 LET A = 0 is executed:

(1) Find the number in memory space A (ACTION: Find; OBJECT: Number; LOCATION: Memory).

(2) Erase the number in memory space A (ACTION: Erase; OBJECT: Number; LOCATION: Memory).

(3) Find the number indicated on the right of the equal sign (ACTION: Find; OBJECT: Number; LOCATION: Statement).

(4) Write this number in memory space A (ACTION: Write; OBJECT: Number; LOCATION: Memory).

(5) Find the next statement in the program (ACTION: Find; OBJECT: Statement; LOCATION: Program).

Experts and novices are likely to differ with respect to their mental models for a programming language. For example, an expert programmer may have developed an accurate conception for a counter set LET, such as the one given above. That is, the expert knows that the value in memory space A is replaced by 0. However, the novice may lack a coherent mental model or may possess incorrect ones for BASIC statements.

In a recent study, Mayer & Bayman (1981) asked novice and expert calculator users to predict the answer that would be displayed for a series of problems such as 2+3+ or 2+++=. Subjects' responses were matched to

transactional expressions for each button press. Results indicated that subjects differed greatly in their conceptions of "what goes on inside the calculator" for each key press, with experts possessing more sophisticated conceptions than novices. Thus, although all subjects were able to use calculators to solve basic math problems--i.e. all users had acquired the basic information for performance--they differed greatly in how sophisticated a mental model they possessed. This work with calculators suggests that learning of the basic information about how to use a language does not guarantee that a user has also acquired a useful mental model for the language.

Moran (1981) suggests that the user develops a "conceptual model" of the system as he or she learns and uses it. He defines the user's conceptual model as the knowledge that organizes how the system works and how it can be used to accomplish tasks. How much training one needs to acquire a conceptual model has not been explored yet. Studying novices' understanding of the programming statements constitutes the first step in addressing this issue. In this paper, we describe the novice programmers' conception of elementary BASIC statements, using a transaction analysis. There is yet no empirical evidence on novices' mental models of the statements of BASIC, at this level of detail.

Method

The goal of this study is to assess the nature of novice programmers' mental models for BASIC statements, following preliminary instruction in BASIC. In particular, this study assesses the transactions that each subject attributes to each of nine BASIC statements, following instruction in how to use these statements. Some transactions are essential for understanding a statement. We will determine how many subject show evidence of having

acquired these essential transactions for each statement. Some transactions are incorrect, such as thinking that READ A involves printing out a number. We will determine how many subjects show evidence of incorrect transactions for each statement. The final product of this study will be a frequency table for each statement, showing how many subjects appeared to possess each major transaction.

Subjects

The subjects were 30 undergraduates at the University of California, Santa Barbara who had no prior knowledge about computer programming. Participation in the study was partial fulfillment of the requirements for the introductory psychology course that they were taking.

Materials and Apparatus

Questionnaire. The questionnaire was an 8½x11 sheet of paper consisting of typed questions regarding: (1) the subject's mathematics background; such as geometry, algebra, and calculus courses the individual had taken in high school and in college, (2) the subject's demographic characteristics; such as academic major, age, sex, GPA and SAT-Verbal and SAT-Math scores, and (3) the subject's computer programming background, such as whether the individual knew how to program a calculator and whether the individual had ever typed at a computer terminal.

IBM Computer Programmer Aptitude Test. This pretest consisted of five 8½x11 inch sheets of paper based on a shortened version of the three-part IBM Computer Programmer Aptitude Sample Test (Luftig, 1980). Part I (PAT-1) consisted of 25 number series problems, with a 5-minute time limit. Part II (PAT-2) consisted of 18 picture analogies, with an 8-minute time limit. Part III (PAT-3) consisted of 12 arithmetic story problems, with an 8-minute time limit.

Revised Minnesota Paper Form Board Test (Series AA). This pretest consisted of 64 spatial visualization problems, with an 8-minute time limit.

Lessons. Both Lesson I and Lesson II were composed from a self-instruction, self-paced, mastery text called BASIC in Six Hours (Marcus, 1980) that is widely used in teaching BASIC in the Microcomputer Laboratory of the University of California, Santa Barbara. The text required that the user has hands-on access to an Apple-II Computer. The text did include mastery tests at the end of each lesson but did not provide any conceptual models to explain the statements.

Lesson I. This eleven page typewritten lesson covered statement execution in the immediate mode. The statements taught were the PRINT, LET, and IF-THEN statements. There were exercises for doing simple arithmetic calculations, for having character strings printed on the display screen, and for assigning values to numerical and character string variables. The use of the semicolons and commas for spacing on the screen, the use of the colon for statement stacking and the use of the command NEW were taught in conjunction with the statements. The lesson ended with a 10-minute self test that covered the essential information taught in the booklet.

Lesson II. This fourteen page, typed lesson covered program preparation, adding or deleting statements in a program, obtaining the list of statements on the display screen, and how to run a program. Also the lesson covered the use of the GOTO statements, the IF statement, the numerical and character string INPUT statements, and the READ and DATA statements. Several sample programs were included, and there was a 10-minute self test at the end of the lesson.

Posttests. The study involved two posttests--verbal test and visual test--with the same nine statements tested in each.[2]

Verbal posttest. This booklet contained 8½x11 inch sheets of paper, with a single statement tested on each sheet. The statements in the test were (in order of presentation):

```
LET A = B + 1
PRINT C
LET D = 0
PRINT "C"
IF A < B GOTO 99
INPUT A
20 DATA 80, 90, 99
30 READ A
60 GOTO 30
```

The immediate mode statements were presented one on a sheet, with a carriage return (indicated by <CR>) after each; the line numbered statements were presented within a simple program. For both, subjects were instructed to write, in plain English, the steps that the computer would carry out for each statement. They were instructed to write each step on a separate line of the test sheet. Subjects were instructed to work on the statements in the given order and were not allowed to skip ahead or go back to the previous page.

Visual Test. This test involved the same nine statements as in the verbal test, presented in the same order and format. In addition, this test began with a three page typed introduction, which presented a visual diagram of the computer. The diagram was based on earlier experiments (Mayer, 1981) and consisted of four parts--the display screen with keyboard, the memory, the input system, and the control system. An example of the diagram is given in Figure 1. For each of the nine statements, subjects were asked to indicate the state of each of the four components of the computer after a

particular statement was executed. For each statement there was a diagram that showed the state of the computer before the statement was executed and another diagram that was left blank for the subject to fill in.

Figure 1 about here

Apparatus. Apparatus consisted of four Apple II Plus computers, with 48 K of memory, and with Amdek 12" Model 100 B/W monitors.

Procedure

The study consisted of six sessions. In the first session, subjects in groups of up to six filled out the questionnaire, received a brief description of the study, signed up for times for the next five sessions, and took the series of the four pretests. Tests were administered in the order, PAT-1, PAT-2, PAT-3 and Minnesota Paper Form Board Test, with time limits of 5, 8, 8, and 8 minutes, respectively. In the following three sessions each subject worked on the two lessons at his or her own pace. Subjects worked individually at the Microcomputer Laboratory,[1] using Apple II computers in conjunction with the text. Following each lesson, subjects' performance on the self test was checked, they were asked to work on the questions they missed until they passed the test. In the final two sessions subjects took the posttests at their own pace, in groups of no more than five people at one time. The verbal test was always given before the visual test.

Results

Scoring

Each subject's protocol for each of the nine statements on the verbal test was scored by two judges. Each protocol was broken down into a list of transactions. A transaction expresses some action performed on some object

at some location in the computer (see Mayer, 1979). For example, one subject's protocol for LET A = B + 1 was:

1. Prints what is typed on screen
2. Understands what the word LET means
3. Solves the equation with information given
4. Gives a solution to problem

These steps can be translated into the following transactions:

PRINT the equation on the screen.

FIND the statement that was just entered.

SOLVE the equation that was just entered.

PRINT the solution on the screen.

Disagreements between judges were rare and were settled by consensus.

Each subject's diagram for each of the nine statements in the visual test was scored by one judge.[3] The judge noted the specific contents of each of the four memory spaces, the specific contents of the display screen, the contents of the input system, and the direction of the arrow in the control system. (Note that executive control of order of line execution could not be determined in this test--so key aspects of IF and GOTO statements could not be scored.) Each subject's diagram was converted into a list of transactions by comparing the initial state of each component and the state of each component following statement execution as produced by the subject. For example, one subject fills in the components of the computer diagram for LET A = B + 1 as shown in Figure 2. As can be seen, the four memory spaces contain A = B + 1, 0, 21, 15, respectively (as compared to the initial contents of 0, 21, 15, 66, respectively), the display screen is intact showing READY followed by the cursor, the input system contains 5, 99, 6, and 7 in each case and the control arrow points to WAIT (as compared to RUN in

the initial diagram). The transactions indicated by this subject are:

For the memory spaces --- Write A = B + 1 in memory space A.
Move the numbers in each space to the next memory space.

For the control system --- Wait for the next statement to be entered from the keyboard.

Figure 2 about here

For each of the nine statements a list of correct transactions was generated, based on an earlier analysis of Mayer (1979). The correct transactions represent the events which are essential for describing the execution of the statement. For example, the correct transactions for LET A = B + 1 are as follows:

1. Find the number in memory space A.
2. Erase that number.
3. Find the number in memory space B.
4. Add one to that number.
5. Write the obtained value in memory space A.
6. Find the next statement that is entered at the keyboard.

The key transaction(s) for each statement were determined by two judges, based on the most characteristic event(s) in the statement. For example, the key transaction for LET A = B + 1 is, "Write the obtained value in memory space A."

For each statement, a list of alternative transactions was also generated, based on the subjects' answers. Some of the transactions were unnecessary, such as printing the statement on the screen. Some of the

transactions were incomplete, such as writing B + 1 in memory space A. Incomplete versions of key transactions, such as the previous example, were also identified. Some of the transactions were incorrect, such as writing A = B + 1 in memory or printing the value of A on the screen. The incorrect transactions represent students' misconceptions of the events involved for a statement.

Frequency of Misconceptions

For each of the nine statements, a frequency table was generated by tallying each subject's answer on the verbal test and on the visual test against the list of all possible correct, unnecessary, incomplete and incorrect transactions. Tables 1 through 9 presents summaries of the data for each of the nine statements, respectively. Hence, each table lists only the major transactions for a given statement,[4] and shows the proportion of subjects who produced each transaction on the verbal test and on the visual test. Proportions are reported separately for the verbal and visual tests. The missing proportions for a transaction on either test indicates that the transaction could not be detected from that test. For example, transfer of control from one line to another cannot be detected from the visual test. The transactions in the tables are stated in plain English. Key transactions are indicated by double asterisks(**).

Tables 1 through 9 about here

Note that on the verbal test subjects are asked to write the steps that the computer goes through in executing the programming statement. In contrast, on the visual test subjects are asked to fill in the four-component

computer diagram (as shown in Figure 2) indicating the changes that occur due to the execution of the statement. The following expository discussion of results is based on the subjects' answers in both of the tests but focuses on the data from the verbal test. This is done because the verbal test allows a more detailed description of subjects' answers than the visual test, the verbal test was administered before the visual test, and the visual test did not provide information on transfer of control.

LET A=B+1. The major misconceptions for this statement can be grouped into three categories: (1) That the computer writes A=B+1 in memory or in memory space A. Forty-seven percent of the subjects thought that the computer stored the equation instead of the value obtained from B+1. (2) That the computer prints the equation or A or the value of A on the screen. Twenty-three percent of the subjects answered this way. (3) That the computer solves the equation A=B+1. Thirteen percent of the subjects believed that this is the case.

Only 30% of the subjects' answers included the key transaction that the computer would store the value from B+1 for the variable A.

Table 1 gives the specific proportions produced by subjects' answers of the major incorrect and incomplete transactions as well as of the set of correct transactions.

(2) LET D=0. The major misconceptions for this statement can be grouped into three categories: (1) That the computer writes the equation in memory. Forty-seven percent of the subjects opted for this idea. (2) That the computer solves the equation. Seven percent of the subjects answered this way. (3) That the computer prints the equation on the screen. Seven percent of the subjects thought this might happen.

The key transaction that the computer writes the value zero in its

memory for the variable D is given by 47 percent of the subjects.

For more detailed description of the subjects' answers refer to Table 2 which includes proportions produced for an incomplete transaction and for the set of correct transactions besides proportions for the major incorrect transactions described above.

(3) PRINT C. The major misconceptions for this statement can also be grouped into three categories: (1) That the computer prints the letter C on the screen. Thirty-three percent of the subjects incorrectly answered this way. (2) That the computer prints either error or nothing on the screen. Seven percent of the subjects had this particular conception. (3) That the computer writes C in its memory. Seven percent of the subjects held this idea.

The key transaction that the computer would print the value of the variable C on the screen was given by 40 percent of the subject population.

See Table 3 for more detailed classification of the subjects' answers for this statement.

(4) PRINT "C". Three major misconceptions for this statement are: (1) That the computer prints the value of the variable C on the screen. Seven percent of the subjects answered this way. (2) That the computer writes C in its memory. Again 7 percent of the subjects thought this was the case. (3) That the computer finds the number in memory space C. Only one subject answered this way.

Eighty-three percent of the subject population stated the key transaction that the computer would print C on the screen.

See Table 4 for more detailed information on the subjects' answers for this statement.

(5) INPUT A. The major misconceptions for this statement are: (1) That

the computer writes A in a data list or memory. Thirty percent of the subjects' answers fell in this category. (2) That the data or A will be printed on the screen. Only one subject held this conception.

There are three key transactions for the INPUT statement: (1) That the computer prints a question mark on the screen. Seven percent of the subject population stated this transaction. (2) That the computer waits for the value of A and <CR> to be entered from the keyboard. Twenty-three percent of the subjects' answers implied this conception. (3) That the computer stores the entered value in memory space A. Only one subject held this conception.

See Table 5 for the proportions produced by subjects' answers for the major transactions for this statement.

(6) IF A<B GOTO 99. The major misconceptions can be grouped into four categories: (1) That the computer prints number 99 or line 99 or an error on the screen. Twenty percent of the subject population believed this was the case. (2) That the computer finds number 99 if A is less than B. Thirteen percent of the subjects answered this way. (3) That the computer writes A or B or A is less than B in memory. Ten percent of the subjects' answers fell in this category. (4) That the computer moves to line 99 without any test of whether the condition A<B is true. Ten percent of the subjects thought that this was the case.

There are two key transactions for the conditional GOTO statement: (1) That execution would move to line 99 in the program if the value of A is less than the value of B. Sixty-three of the subjects held this conception. (2) That execution would continue with the next statement in the program if the value of A is not less than the value of B. Only 33 percent of the subject population stated this conception in their protocols.

For more detailed list of the transactions and the corresponding proportions of subjects' answers see Table 6.

(7) 20 DATA 80, 90, 99. The major misconception for the DATA statement is that the computer prints the numbers on the screen. Thirteen percent of the subject population answered this way.

Only 27 percent of the subjects gave the precise answer that the numbers 80, 90, 99 would be put in memory or the input queue.

See Table 7 for more detailed categorization of the subjects' answers.

(8) 30 READ A. The major misconceptions for the READ statement can be grouped into three categories: (1) That the computer prints the value of A on the screen. Ten percent of the subjects believed in this idea. (2) That the computer writes A in memory. One subject stated this conception. (3) That the computer waits for a value to be entered from the keyboard. Again, only one subject held this wrong conception.

Only 10 percent of the subject population answered that the first date value from the DATA statement would be written in memory space A.

See Table 8 for more detailed description of the subjects' answers for the READ statement.

(9) 60 GOTO 30. The major misconceptions are: (1) That the computer finds 30 if A is not equal to some number. Seven percent of the subjects answered this way. (2) That the computer prints line 30 on the screen. Only one subject gave this answer.

There are two key transactions for the simple GOTO statement: (1) That the computer moves program execution to line 30 in the program. Sixty-seven percent of the subject population held this conception. (2) That the computer continues with program execution from that line. Only 37 percent of the subjects' answers implied that they had this idea.

Table 9 lists these transactions and proportions corresponding to each transaction.

Differences among statements. In order to make comparisons of subjects' conceptions among the nine statements, each subject's verbal protocol for each statement was categorized as correct, incomplete, empty, or incorrect. The criteria for classifying protocols were: (1) correct--if the subject's protocol included the key transaction(s) and no incorrect transactions, (2) incomplete--if the subject produced one or more incomplete versions of the key transaction(s) and no incorrect transactions, (3) incorrect--if the subject produced one or more incorrect transactions, and (4) empty--if the subject produced no key transactions, no incomplete version of a correct transaction, and no incorrect transaction. Table 10 presents a summary of the proportion of users producing each type of conception for each of the nine BASIC statements.

Table 10 presents the nine statements in order of difficulty based on proportion correct conceptions. As can be seen, PRINT "C" is the best comprehended statement--with 80% of the subjects expressing the correct conception--while INPUT A is the worst understood statement--with only one subject indicating a correct conception. It should also be noted that substantial numbers of subjects hold "empty" conceptions for the READ, DATA, INPUT, and PRINT C statements.

Table 10 about here

A corresponding summary table based on the visual test was also prepared. Table 11 presents seven of the nine statements in order of difficulty based on proportion correct statements for this test. Data for the IF and GOTO statements has been excluded because the visual test did not allow for expressing transfer of control adequately. Although there seems to

be general similarity in the patterns of performance between the two tests, making specific comparisons between them is not appropriate because of the differences in the nature of tests. Note that the verbal test allowed subjects to exhibit incomplete and empty conceptions; in contrast, the visual test with its four-part computer diagram forced subjects to be specific.

Table 11 about here

Pretests. An additional analysis was performed in order to determine whether performance on the pretests was related to development of the correct conceptions of the BASIC statements. First, correlations were run between each pretest score (MPFBT, PAT-1, PAT-2, and PAT-3) and the total number of correct conceptions for the nine BASIC statements. Table 12 shows the correlation matrix, with significant r values marked with an asterisk(*). As can be seen, only PAT-2--a spatial reasoning test--seems to be significantly related to having correct conceptions for the BASIC statements ($r = .43$, $p < .02$). Also, the two spatial reasoning tests, MPFBT and Pat-2, show a significant correlation ($r = .49$, $p < .01$) as well as PAT-2 and PAT-3 ($r = .39$, $p < .05$). In addition, a stepwise regression analysis was performed with each of the four pretest scores as the independent variables and the total number of correct conceptions of statements as the dependent variable. The resulting equation selected only one variable--PAT-2--and accounted for 18 percent of the variance in the dependent measure. These analyses suggest that the PAT-2 test was the only test that was related to understanding of the BASIC statements, although the relationship is rather mild.

Table 12 about here

Conclusions and Recommendations

In this study we attempted to evaluate the mental models acquired by novices following several hours of learning and practice in BASIC. It is important to note that we focused on novices' learning of mental models rather than learning of specific information. In order to evaluate mental models, we adopted a detailed level of analysis--transaction analysis (Mayer, 1979). This analysis has allowed us to identify the misconceptions and missing conceptions in novices' understanding of the conceptual models underlying elementary BASIC statements.

The results of the study indicate that mastery of the information in the instructional booklet does not guarantee that users will acquire useful mental models for how the BASIC statements function. Although all of the subjects in this study were able to adequately answer the questions on the mastery test at the end of each lesson, only about one-third of the subjects were able to produce correct conceptual descriptions of an average statement such as PRINT C or LET A=B+1 and more than one-half of the subjects generated incorrect or empty conceptual descriptions.

Not surprisingly, the nature of the misconceptions are found to differ from statement to statement; some statements being more difficult to grasp than others. The programming statements in order of difficulty (based on the verbal test) can be listed as: INPUT A, 30 READ A, IF A<B GOTO 99, LET A = B + 1, 20 DATA 80, 90, 99, 60 GOTO 30, PRINT C, LET D = 0, and PRINT "C". However, this part of the results should not be taken as conclusive since we not only used a selective number of programming statements but

presented them with a specific order in the testing sessions. One needs to compose an exhaustive list of the programming statements of BASIC and use different presentation orders to be able to make conclusive remarks about the relative comprehensibility of each statement.

In this paper, we analyzed beginning programmers'as a group so that we could focus on the most common misconceptions. There seems to be sufficient evidence for specific recommendations to be made for each statement included in this study.

1. Recommendations concerning INPUT. Subjects have difficulty in conceiving where the to-be-input data comes from and how it is stored in memory. Many subjects fail to understand the nature of executive control--i.e. that the computer will "wait" for input from the keyboard. These learners need explicit training concerning the role of input terminal, the wait-run control, and the memory spaces--including visual representations, verbal descriptions of key transactions (as listed in Table 5), and role playing by the learner.

2. Recommendations concerning READ-DATA statements. Subjects have difficulty in conceiving where the to-be-read data comes from and how it is stored in memory. Subjects need explicit training concerning the data stack and the memory spaces--including visual representations, verbal descriptions of the key transactions (as listed in Tables 7 and 8), and role playing by the learner.

3. Recommendations concerning the conditional GOTO and simple GOTO. Subjects' major difficulty with the GOTO is that they cannot conceive of what will happen next after program execution moves on to the desired line. Also, with the conditional GOTO, they seem to have difficulty in conceiving what would happen next if the condition is false. Hence, beginners need training

at the terminal to observe execution control, explicit training with verbal descriptions of the key transactions (as listed in Tables 6 and 9) and role playing to get a feel for execution control.

4. Recommendations concerning LET statements. Subjects seem to get confused between solving an equation (i.e. treating the equal sign as an equality) and making an assignment. Those who seem to understand the assignment property in the statement still have difficulties in conceiving where to store the assigned values. Beginning learners need explicit training concerning the specific memory locations and under what conditions values stored in those locations get replaced. Visual representations, verbal descriptions consisting of the set of correct transactions (as listed in Tables 1 and 2), and role playing by the learner are ways to overcome beginners difficulties.

5. Recommendations concerning the PRINT statements. Subjects seem to confuse the function of PRINT C and PRINT "C". Also, subjects have difficulty in conceiving that these statements simply display on the screen what is asked to be printed; they incorrectly assume that the computer keeps a record of what is printed somewhere in its memory. Beginning programmers need explicit comparative training for the two types of PRINT statements. Training can be at the terminal so that learners observe the differences in output; training can include visual representations for memory spaces showing no change after statement execution, verbal descriptions of the key transactions (as listed in Tables 3 and 4), and role playing by the learner.

The present study has focused on diagnosis of bugs in novices' mental models for BASIC statements. The specific diagnosis of what users do not know--at the level of missing or incorrect transactions--allows us to develop individual instructional techniques for remediation. Thus, the next logical

step in this project is to determine whether instructional techniques can be developed to correct users' mental models for BASIC.

References

Luftig, M. Computer programmer aptitude tests. New York: Arco, 1980.

Marcus, J. Basic in six hours: A self-instruction text. Santa Barbara, CA: Microcomputer Laboratory, 1980.

Mayer, R. E. A psychology of learning BASIC. Communications of the ACM, 1979, 11, 589-593.

Mayer, R. E. The psychology of how novices learn computer programming. Computing Surveys, 1981, 1, 121-141.

Mayer, R. E., and Bayman, P. Psychology of calculator languages: A framework for describing differences in users' knowledge. Communications of the ACM, 1981, 24, 511-520.

Moran, T. P. An applied psychology of the user. Computing Surveys, 1981, 1, 1-11.

Footnotes

This project was supported by Grant NIE-G80-0118 from the National Institute of Education, Program in Teaching and Learning. Richard Welker assisted in data tabulation. Requests for reprints should be sent to: Piraye Bayman or Richard E. Mayer, Department of Psychology, University of California, Santa Barbara, CA 93106.

[1]We wish to thank Jeffrey Marcus and the staff of the Microcomputer Laboratory at the University of California, Santa Barbara, for their assistance on this project.

[2]The test booklets used in this study included additional questions that were not analyzed, and do not influence the present results.

[3]Since scoring was literal, there was no need for an additional judge.

[4]Some unnecessary, incomplete and incorrect transactions are excluded for purpose of clarity. However, detailed versions of these tables can be obtained from the authors upon request.

Table 1

Proportion of Users Who Produced Each Transaction for LET A=B+1

Verbal Test	Visual Test	
		Incorrect Transactions
.43	.17	Write A=B+1 in memory or in memory space A.
.23	.17	Print A=B+1 or A or the value of A on the screen.
.13	.17	Solve the equation in the statement.
		Incomplete Transaction
.33	.20	Write B+1 in memory space A.
		Set of Correct Transactions
.20	---	Find the number in memory space A.
.03	---	Erase the number in memory space A.
.03	---	Find the number in memory space B.
.03	---	Add 1 to the number in memory space B.
.30	.33	** Write the obtained value in memory space A.
.27	.60	Find (wait for) the next statement to be entered in the keyboard.

Note. - Double asterisk(**) indicates the key correct transaction.

Dash (---) indicates transaction could not be evaluated from the test.

Table 2

Proportion of Users Who Produced Each Transaction for LET D=O

Verbal Test	Visual Test	
		Incorrect Transactions
.47	.10	Write the equation in memory.
.07	---	Solve the equation in the statement.
.07	.13	Print the equation on the screen.
		Incomplete Transaction
.13	.03	Write D or O in memory.
		Set of Correct Transactions
.07	---	Find the number in memory space D.
.03	---	Erase the number in memory space D.
.47	.77	**Write O in memory space D.
.17	---	Find the next statement to be entered in the keyboard.

Note. --Double asterisk(**) indicates the key correct transaction.
Dash (---) indicates transaction could not be evaluated from the test.

Table 3

Proportion of Users Who Produced Each Transaction for PRINT C

Verbal Test	Visual Test		
			Incorrect Transactions
.33	.10		Print the letter C on the screen.
.07	---		Print either error or nothing on the screen.
.07	.17		Write C in memory.
			Incomplete Transaction
.03	---		Print
			Set of Correct Transactions
.17	---		Find the number in memory space C.
.40	.60	**	Print the number or zero on the screen.
.10	---		Find the next statement to be entered in the keyboard.

Note.-- Double asterisk (**) indicates the key correct transaction.
Dash (---) indicates transaction could not be evaluated from the test.

Table 4

Proportion of Users Who Produced Each Transaction for PRINT "C"

Verbal Test	Visual Test	
		Incorrect Transactions
.07	.17	Print the value of C on the screen
.07	.03	Write C in memory
.03	---	Find the number in memory space C
		Incomplete Transaction
.07	---	Do not print the value of C on the screen.
		Set of Correct Transactions
.07	---	Find the letter C in quotes in the statement.
.83	.47	**Print C on the screen.
.10	---	Find the next statement to be entered in the keyboard.

Note. -- Double asterisk (**) indicates the key correct transaction. Dash (---) indicates transaction could not be evaluated from the test.

Table 5

Proportion of Users who Produced Each Transaction for INPUT A

Verbal Test	Visual Test	
		Incorrect Transaction
.30	.17	Write A in memory or data list.
.03	.20	Print the data or A on the screen.
		Incomplete Transactions
.13	---	Wait for some data or number or A.
.07	.10	Write a number.
		Set of Correct Transactions
.07	.10	** Print ? on the screen
.23	---	** Wait for the number and the <CR> to be entered from the keyboard.
.00	---	Find the number entered in the keyboard.
.03	---	Find the number in memory space A.
.03	---	Erase the number in memory space A.
.03	---	** Write the number just entered in memory space A.
.10	---	Find the next statement to be entered in the keyboard.

Note. - Double asterisk (**) indicates the key correct transaction.
Dash (---) indicates transaction could not be evaluated from the test.

Table 6

Proportion of Users Who Produced Each Transaction for IF A<B GOTO 99

Verbal Test	Visual Test	
		Incorrect Transactions
.20	.13	Print number 99, line 99, or error or the screen.
.13	---	If A is less than B, then find number 99
.10	.03	Write A or B or A is less than B in memory.
.10	---	Find line 99 in the program
		Set of Correct Transactions
.33	---	Find the number in memory space A.
.33	---	Find the number in memory space B.
.43	---	Test if the number in memory space A is less than the number in memory space B.
.63	---	** If the value of A is less than the value in B, then move to line 99 in the program.
.33	---	** If the value of A is not less than the value of B, then move on to the next statement in the program.
.33	---	Continue with the execution of the program from there.

Note. -- Double asterisk (**) indicates the key correct transaction.
Dash (---) indicates transacrion could not be evaluated from the test.

Table 7

Proportion of Users Who Produced Each Transaction for 20 DATA 80, 90, 99

Verbal Test	Visual Test	
		Incorrect Transaction
.13	.13	Print the numbers on the screen.
		Incomplete Transaction
.27	.30	Put the data in memory space A or in memory
		Set of Correct Transactions
.03	---	Find the numbers in the statement
.27	.60	**Put the numbers in memory or in the input queue.
.00	---	Find the next statement in the program.

Note. --Double asterisk (**) indicates the key correct transaction.

Dash (---) indicates transaction could not be evaluated from the test.

Table 8

Proportion of Users Who Produced Each Transaction for READ A

Verbal Test	Visual Test	
		Incorrect Transactions
.10	.07	Print the value of A on the screen.
.03	.07	Write A in memory.
.03	---	Wait for a value to be entered from the keyboard.
		Set of Correct Transactions
.07	---	Find the DATA statement in the program.
.07	---	Find the number in memory space A.
.00	---	Erase the number in memory space A.
.10	.13	**Write the first number from the DATA statement in memory space A.
.10	---	Find the next statement in the program

Note. -- Double asterisk (**) indicates the key correct transaction.
Dash (---) indicates transaction could not be evaluated from the test.

Table 9

Proportion of Users Who Produced Each Transaction for 60 GOTO 30

Verbal Test	Visual Test	
		Incorrect Transactions
.07	---	If A does not equal to X or 99 find 30.
.03	.10	Print line 30 on the screen.
		Set of Correct Transactions
.00	---	Find the line number in the statement.
.67	---	**Move to line 30 in the program.
.37	---	**Continue with program execution from that line.

Note. --Double asterisk (**) indicates the key correct transaction.

Dash (---) indicates transaction could not be evaluated from the test.

Table 10

Proportion of Users with Correct, Incomplete, Incorrect, and Empty Conceptions for the Nine BASIC Statements from the Verbal Test

	Conceptions			
Statements	Correct	Incomplete	Incorrect	Empty
INPUT A	.03	.30	.30	.37
30 READ A	.10	.27	.17	.47
IF A < B GOTO 99	.27	.27	.40	.07
LET A = B + 1	.27	.10	.60	.03
20 DATA 80, 90, 99	.27	.17	.13	.43
60 GOTO 30	.27	.56	.10	.07
PRINT C	.33	.00	.47	.20
LET D = 0	.43	.03	.53	.00
PRINT "C"	.80	.00	.13	.07

Table 11

Proportion of Users with Correct, Incomplete, Incorrect, and Empty Conceptions for the Seven BASIC Statements from the Visual Test

	Conceptions			
Statements	Correct	Incomplete	Incorrect	Empty
INPUT A	.00	.00	.77	.23
30 READ A	.13	.07	.43	.37
LET A = B + 1	.30	.20	.43	.07
PRINT "C"	.37	.00	.53	.10
PRINT C	.50	.00	.47	.03
LET D = 0	.60	.07	.30	.03
20 DATA 80, 90, 99	.60	.17	.17	.07

Table 12

The Correlation Matrix for Pretest Scores and the

Number of Correct Conceptions for the Nine BASIC Statements

	MPFBT	PAT-1	PAT-2	PAT-3	Total Correct
MPFBT	1.00				
PAT-1	0.33	1.00			
PAT-2	0.49*	0.22	1.00		
PAT-3	0.05	-0.03	0.39*	1.00	
Total Correct	0.24	0.14	0.43*	0.04	1.00

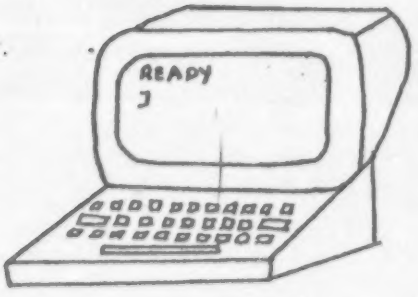
Note. Asterisk (*) indicates $p < .05$.

Figure Captions

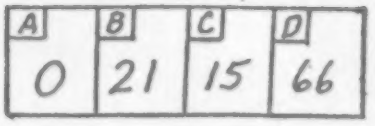
Figure 1. Diagram of the computer used in the visual test.

Figure 2. One subject's changes of the contents of the components of the computer diagram for LET A=B+1.

The display screen and keyboard:



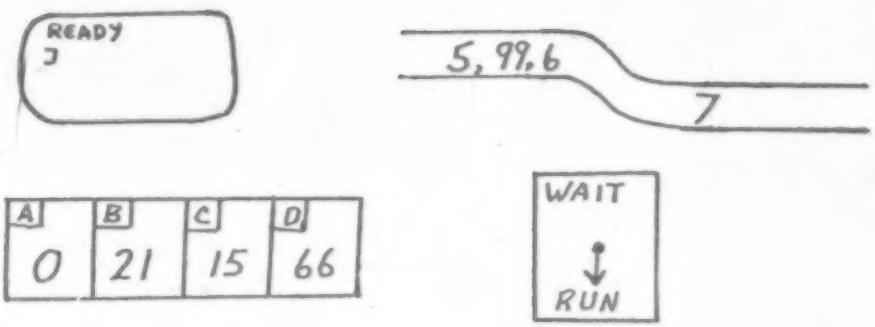
The memory:



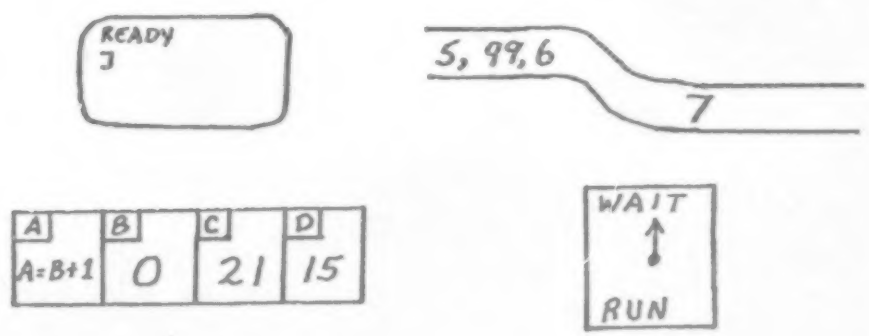
The input system:

The control system:

Initial state:



Subject fills in the components as:



CHAPTER 5

EFFECTS OF INSTRUCTION ON REMEDIATION OF BUGS AND PROBLEM SOLVING FOR CALCULATOR LANGUAGE

Note

This article has been submitted for publication as follows:

Bayman, P., & Mayer, R. E. Instructional manipulation of users' mental models for calculators. International Journal of Man Machine Studies, under review.

Abstract

A mental model for a calculator or computer refers to the user's conception of the "invisible" information processing states and transformations that occur between input and output. This paper explores the following ideas concerning users' mental models for electronic calculators: (1) users differ greatly in their mental models in spite of similar "hands-on" experience, (2) users often tend to develop either impoverished or incorrect models, (3) users can be encouraged to develop more useful mental models through instructional intervention.

Key Words and Phrases: calculator, instruction, learning, psychology

CR Categories: 1.50, 2.12, 4.20

1. Mental Models

Work on human factors related to computers--or what Moran (1981) calls "an applied psychology of the user"--has been drawing increased attention. Within the past year, special issues of the International Journal of Man-Machine Studies (Volume 15, Number 1, July 1981) and of Computing Surveys (Volume 13, Number 1, March 1981) have been devoted entirely to this topic, and the IEEE Computer Society recently commissioned a tutorial text, Human Factors in Software Development, edited by Bill Curtis (1981). An emerging theme from human factors work concerns the important role of what Young (1981a, 1981b) calls the user's "mental model" or what Moran (1981) calls the "user's conceptual model" of the to-be-learned system.

In the course of learning to use a new computer language or a new calculator language, the user may develop a "mental model" of the machine. A mental model is a metaphor consisting of the components and the operating rules of the machine. A mental model of a calculator allows the user to conceive of "invisible" information processing states (such as contents of internal registers) and "invisible" transformations (such as moving the contents of one register into another) that occur between input and output. Mental models are particularly important when users will have to guess the current state of the machine, or have to make inferences about how a given sequence of commands produces a certain output.

Examples of mental models have been proposed for calculator languages (Mayer & Bayman, 1981; Young, 1981a, 1981b), text editing languages (Card, Moran & Newell, 1980; Moran, 1981), file management languages (Mayer, 1980, 1981; Reisner, 1981), LOGO (DuBoulay & O'Shea, 1978; DuBoulay, O'Shea & Monk, 1981), and BASIC (Mayer, 1979, 1981). For example, Young (1981a, 1981b) suggests a simple "register model" for a four function calculator. The components

are: a keyboard for entering numbers, operators, and equals keys; a display for showing one number; an internal number register for holding one number; and an internal operator display for holding one operator. The operating rules include the following: the first number that is entered is stored in the internal number register; for example, the sequence 2+3= will result in 2 being stored in the internal number register. The last operator symbol that is entered will be stored in the internal operator register; for example, the sequence 2-+3= will have the same effect as 2+3= because for both sequences '+' is the last operator entered. The last number that is entered is stored in the display; for example, 2+3 will result in 3 being shown in the display. When the equal key is pressed, or when an operator key is pressed after an expression has been entered, the operation will be carried and the result shown in the display; for example, 2+3= will have the same effect on the display as 2+3+. If an equal key is pressed after an operator key, the operation will be carried out using the number in the internal number register as both the first and second operand; for example, 2+= would result in the display of 4, since '2' is added to itself. This relatively simple model--consisting of a few components and operating rules--is able to generate answers for any sequence of key presses.

2. Useful Mental Models

Users' mental models may vary along several dimensions. (1) Usefulness. A mental model is useful to the extent that it supports sophisticated performance by the user. (2) Completeness. A mental model is complete to the extent that all details concerning the components and operating rules are spelled out. (3) Veridicality. A mental model is veridical to the extent

that the components match the actual physical design of the machine and the operating rules match the machine language. The present study is based on the idea that the first dimension is important for users while the second and third are important for electronics experts.

Users who acquire mental models through "hands-on" experience are apt to develop models that are sufficient for simple computation, but that lack sophistication required for complex mathematical work. Similarly, most calculator manuals focus on procedures for solving simple computational problems, but make no attempt to help the user generate useful conceptual models. In other words, learning to use a calculator through hands-on experience or manual reading may not produce "useful" mental models. Although a user's mental model may be internally self-consistent and appropriate for standard computations, different users may develop quite different models, varying in usefulness. (1) Incorrect performance. A user may possess a model that is consistent with simple expressions of the form, 2+3=, but which generates incorrect performance when extrapolated beyond simple expressions. For example, Mayer & Bayman (1981) found that many users assume that an expression is evaluated as soon as a number key is pressed, such as predicting that 2+3 results in 5 appearing in the display. (2) Impoverished performance. A user may possess a model that is consistent with simple expressions of the form, 2+3=, but which does not always generate correct performance when extrapolated beyond simple expressions. For example, Mayer & Bayman (1981) found that many users assume that an expression is evaluated only when an equals key is pressed. (3) Sophisticated performance. A user may possess a model that is consistent with simple expressions and which also can be usefully extrapolated to complex

expressions. For example, Mayer & Bayman (1981) found that some users assume that an expression is evaluated whenever an operator key (or equals key) is pressed, such as predicting that 2+3+ results in 5 being displayed.

3. Instructional Study

The goal of the present study is to determine whether useful models for calculators can be explicitly taught to users. In this study some users are given modest instructional intervention, namely, introduction of a diagram showing some internal registers. For some users (line group) the diagram presents the internal registers in a line, alternating between "number" and "operator" registers; for other users (stack group) the diagram presents the internal registers as a stack of "number" registers with a single "operator" register to the side; for other users (no model group) no description of internal registers is given. If the introduction of the diagrams encourages users to build more useful mental models, then users who are given the diagrams should show more "sophisticated" performance and less "incorrect" or "impoverished" performance.

3.1 Method

Subjects and design. The subjects were 72 students recruited from Introductory Psychology courses at the University of California, Santa Barbara. Subjects had no previous experience with computer programming, but all were casual users of calculators. Twenty-four subjects served in the line group, 24 served in the stack group, and 24 served in the no model group.

Materials. The materials consisted of a pre-experimental questionnaire, three instruction booklets, and three problem booklets.

The pre-experimental questionnaire was an 8 1/2 x 11 inch sheet of paper

with typed questions. The questionnaire asked about the subject's age, SAT scores, sex, major, year in school, previous mathematics courses, experience with computer programming, ownership of calculators, and experience with calculators.

The instruction booklet for the line group contained three 8 1/2 x 11 inch sheets of typed paper. The first page presented a diagram of the keyboard and display of a typical four function calculator (see Figure 1), and described the digit and operator keys that were relevant to the experiment. The second page presented a diagram and description of the internal registers based on a line model (see Figure 2). The internal registers were represented as a row of alternating lines and boxes, such that a number could be held in each line and an operator in each box. The display was drawn as a rectangle, and was described as that part of the calculator where the answer is shown. The third page provided instructions for the task and gave four sample problems. Each sample problem consisted of a sequence of key presses, such as 3 + 2, and was accompanied by a diagram of the internal line register and display for subjects to fill in. Subjects were told that their job was to fill in what numbers and symbols would be in the internal registers and display after the final key press.

The instruction booklet for the stack group also consisted of three typed 8 1/2 x 11 inch sheets. The first page was identical to the first page of the line group's booklet. The second page presented a diagram and description of the internal registers based on a stack model (see Figure 3). The internal registers were represented as a column of rectangles labeled x-register, y-register, and operator-register. The description pointed out

that the x-register, y-register, and display could hold numbers and that numbers could be transferred from one to the other, while the operator register could hold an operator symbol. As with the line model, the display was represented as a separate rectangle and as the only visible part of the calculator. The final page provided the same task instructions and sample problems as in the line group's booklet, except that the diagram accompanying each sample problem was of the stack model.

Figures 1, 2 and 3 About Here

The instructional booklet for the no model group consisted of two typed 8 1/2 x 11 inch sheets of paper. The first page was identical to the first page for the other booklets; the second page was identical to the last page of the other booklets except that only a display rectangle accompanied each sample problem and the subject's task was to write down what would be in the display after the last key press.

Three problem booklets were constructed using 8 1/2 x 11 inch sheets of paper. Each sheet contained four to seven problems, with each problem consisting of one to seven key strokes. Each key stroke was either a digit (2, 3, or 7), an operation (+ or x), or an equals (=). For the line group, the line model and display were given to the right of each problem; for the stack group, the stack model and display were given to the right of each problem; for the no model group, the display was given to the right of each problem. Examples are given in Figure 4. All booklets contained the same 25 problems in common. Examples are given in Table 1. As can be seen, some

problems involve standard sequences of key presses (standard problems or chain problems) while others involve sequences that violate the grammatical rules of arithmetic (non-standard problems).

Figure 4 and Table 1 About Here

3.2 Results

The number or characters that a subject thought would be in the display was recorded for each of the 25 problems for each subject. Scoring was then broken down into three subsets of the data: (1) Answers to problems 1 through 10 (as partially shown in Table 1) were used to determine the subject's conception of when an expression is evaluated. (2) Answers to problems 11 through 23 (as partially shown in Table 1) were used to determine the subject's conception of how to evaluate a nonstandard sequence. (3) Answers to problems 24 and 25 (as shown in Table 1) were used to determine the subject's conception of the order of execution of operations within a chain.

For problems 1 through 10, earlier studies by Mayer & Bayman (1981) suggest three types of strategies concerning when to evaluate an expression.

(1) Number strategy. According to this strategy, an expression is evaluated as soon as a number key is pressed. Thus, for the problems, 2, 2+, 2+3, and 2+3+, the answers are, respectively, 2, 2, 5, 5. This strategy can be considered "incorrect" because it does not allow for two digit numbers.

(2) Equals strategy. According to this strategy, an expression is evaluated only when the equals key is pressed. Thus, for the problems, 2, 2+, 2+3, and 2+3+, the answers are, respectively, 2, 2, 3, 3. This strategy

can be considered "impoverished" because it is limited to problems in standard algebraic rotation as taught in textbooks.

(3) Operator strategy. According to this strategy, an expression is evaluated when either an operator key (such as + or x) or the equals key (=) is pressed. Thus, for the problems, 2, 2+, 2+3, and 2+3+, the answers are, respectively, 2, 2, 3, 5. This strategy can be considered sophisticated because it goes beyond the normal algebraic notation, and attributes new actions to the calculator. The right panel of Table 1 shows the predicted answers for several of the first 10 problems for each of the three general strategies.

For problems 11 through 23, earlier studies by Mayer & Bayman (1981) suggest three types of strategies concerning how to evaluate a nonstandard sequence.

(1) Ignore strategy. According to this strategy, any sequence of symbols that violate the grammar of arithmetic is ignored. For the problems, 2+=+=+= and 2++3=, the answers would be, respectively, 2 and 5. This strategy can be considered "impoverished" because it mirrors the standard algebraic notation as taught in textbooks.

(2) Reset strategy. According to this strategy, any sequence that violates the grammar of arithmetic will be detected and the display will produce an error message. For the problems, 2+=+=+= and 2++3=, the answers would be, 0 or E or 2 for the first and 0, E or 3 for the second. This strategy is more sophisticated than the preceeding one because it assumes that the calculator can detect and point out errors.

(3) Increment strategy. According to this strategy, any sequence that violates the grammar of arithmetic is subject to incrementation. For the

problems, 2+=+=+= and 2++3=, the answers could be 6 or 8 or 16 for the first and 7 for the second. This strategy is "sophisticated" because it allows for an actively operating calculator that can go beyond the rules of standard textbook notation; the increment strategy suggests that the calculator inserts a number between two consecutive non-numeric symbols. The right panel of Table 1 shows the predicted answers for several of the problems 11 through 23 for each of the three general strategies.

For problems 24 and 25, earlier studies by Mayer & Bayman (1981) suggest three types of strategies concerning the order of execution of operations in a chain.

(1) Incorrect strategies. Several incorrect strategies are possible, including performing the chain from right to left (i.e., with answers of 23 and 20, respectively), or performing additions before multiplications (i.e., with answers of 42 and 20, respectively). These strategies are incorrect because they violate standard algebraic notation and rules as taught in textbooks.

(2) Left-to-right strategy. According to this strategy, the operations are performed in order beginning on the left (i.e., with answers of 35 and 13, respectively). This is an "impoverished" strategy because it is based solely on a knowledge of simple arithmetic notation as taught in textbooks.

(3) Multiplication-before-addition strategy. A more sophisticated strategy, that assumes active processing by the calculator, is that multiplication operations are performed before additions (i.e., 23 and 13, respectively). The right panels of Table 1 show the predicted answers for problems 24 and 25 for each of the three general strategies.

For each subject, a tally was made of the number of answers for problems 1 through 10 that matched answers for each of the three strategies for "when to evaluate" an expression. The best fitting evaluation strategy for each subject was determined by selecting the strategy that most often produced answers matching those given by subjects on problems 1 through 10. Table 2 shows the proportion of subjects in each group who were classified as using number, equals, and operator strategies. As can be seen, there is a trend in which line subjects and stack subjects are more likely to evaluate for an operator (i.e., the sophisticated strategy), than the control group. A chi-square test was conducted on the data, with the number and equals strategies collapsed into one cell. The chi-square test revealed that the three groups differed significantly with respect to the proportion of subjects using a sophisticated strategy, $\chi^2 = 9.01$, df = 2, $p < .02$.

Table 2 About Here

For each subject, a tally was made of the number of answers for problems 11 through 23 that matched answers for each of the three strategies for how to evaluate a non-standard expression. The best fitting strategy was determined for each subject as described above, and Table 3 gives the proportion of subjects from each group that were classified as using the ignore, reset, and increment strategies. As can be seen, the line group and, to some extent, the stack group, tend to be more likely than the control group to use sophisticated strategies such as increment or reset. A chi-square test was conducted on the data in Table 3, with the reset and increment strategies

collapsed into one cell. The chi-square test revealed that the three groups differed significantly with respect to the proportion of subjects using sophisticated strategies, $\chi^2 = 6.20$, df = 2, $p < .05$.

Table 3 About Here

For each subject, a tally was made of the number of answers for problems 24 and 25 corresponding to each of the three chain strategies. The best fitting strategy was determined for each subject as described above, and Table 4 summarizes the proportion of subjects from each group who were classified as using incorrect, left-to-right, and multiply-before-add strategies. As can be seen in the table, there are no major differences among the groups. One reason for the failure to identify any differences among the groups may be that only two problems were used on the test.

Table 4 About Here

---

4. Conclusions and Recommendations

This study provides information concerning three major conclusions.

1. Users differ greatly in their mental models despite similar hands-on experience. For example, in the control group about a third used the number strategy, about a third used the equals strategy, and about a third used the operator strategy for when to evaluate expressions. Presumably, all the subjects were able to competently use their calculators for basic arithmetic computations, but they acquired quite different conceptions of how calculator language operates.

2. Users often develop impoverished or incorrect models. For example, a third of the control subjects and 8% of the model subjects tend to exhibit an incorrect strategy (i.e., "number strategy") for when to evaluate an expression. Thus, there is adequate reason to believe that subjects who learn solely by "hands-on experience" may develop mental models that either are quite limited or outright wrong.

3. Users can be encouraged to develop more useful mental models through instructional intervention. Mayer & Bayman (1981) found subjects who had programming experience (i.e., experts) were more likely to use sophisticated strategies for when to evaluate an expression (i.e., operator strategy) and how to evaluate a non-standard expression (i.e., increment and reset strategies) than non-programmers (i.e., novices). In the present study, the tendency to act like experts was enhanced by teaching casual users to make use of simple models of the internal registers of the calculator. Thus, the present study demonstrates that mental models can be explicitly taught to users of electronic computing devices, and that these models can enhance the level of sophistication of user performance. Of course, it is not argued that such an instructional intervention as used in this study would be most helpful for all nonexpert users; neither is it argued that any one mental model would work for all individuals. Further research is necessary to reveal "good mental models" that would help the majority of individuals to have a better grasp of the operation of the electronic device that they work with.

This study suggests recommendations for how to provide and evaluate instruction for electronic computing devices such as calculators and computers.

1. Provide the novice user with a simple and useful model of the internal components and operating rules. Some models may be more effective than others. For example, the line model may be more useful for calculators using algebraic logic while the stack model may be more useful for calculators using Reverse Polish Notation (RPN).

2. Encourage the user to relate "hands-on experience" or "manual instructions" to the model. Hands-on experience does not guarantee that the user will "understand" what he or she is doing. Thus the user should be challenged to predict what is going on "inside" the device during computations. Similarly, asking a user to learn solely from a manual may also produce unsophisticated mental models, since manuals generally do not discuss internal components and processes. When manuals fail to provide adequate models, instructors should provide the models and encourage users to "translate" the manual with respect to the models.

3. The usefulness of a model should be evaluated. There is no guarantee that all models will increase the sophistication of user performance. One yardstick is to compare novices who use various models (or no model) against experts. Models that increase the tendency to act like experts in a domain are useful; those that do not make users act like experts are not useful and should be replaced.

References

Card, S. K., Moran, T. P., and Newell, A. (1980). Computer text-editing: An information processing analysis of a routine cognitive skill. Cognitive Psychology, 12, 32-74.

Curtis, B. (1981). Human factors in software development. Los Angeles: IEEE Computer Society Press.

DuBoulay, B. & O'Shea, T. (1978). Seeing the works: A strategy for teaching interactive programming. In Proceedings of the Workshop on Computing Skills and Adaptive Systems. Liverpool.

DuBoulay, B., O'Shea, T. and Monk, J. (1981). The black box inside of the glass box: Presenting computing concepts to novices. International Journal of Man-Machine Studies, 14, 237-249.

Mayer, R. E. (1979). A psychology of learning BASIC. Communications of the ACM, 11, 589-593.

Mayer, R. E. (1980). Elaboration techniques and advance organizers that affect technical learning. Journal of Educational Psychology, 72, 770-784.

Mayer, R. E. (1981). The psychology of how novices learn computer programming. Computing Surveys, 13, 121-141.

Mayer, R. E. and Bayman, P. (1981). Psychology of calculator languages: A framework for describing differences in users' knowledge. Communications of the ACM, 24, 511-520.

Moran, T. P. (1981). An applied psychology of the user. Computing Surveys, 13, 1-11.

Moran, T. P. (1981). The command language grammar: A representation for user interface of interactive computer systems. International Journal of Man-Machine Studies, 15, 3-50.

Reisner, P. (1981). Human factors studies of database query languages: A survey and assessment. Computing Surveys, 13, 13-31.

Young, R. M. (1981a). Surrogates and mappings: Two kinds of conceptual models for pocket calculators. Paper presented at Conference on Mental Models, San Diego.

Young, R. M. (1981b). The machine inside the machine: Users' models of pocket calculators. International Journal of Man-Machine Studies, 15, 51-85.

Footnote

This research was supported by Grant NIE-G80-0118 from the National Institute of Education, Program in Teaching and Learning. Richard Welker and Gloria Eurotas assisted in the analysis of data. Requests for reprints should be sent to Richard E. Mayer or Piraye Bayman, Department of Psychology, University of California, Santa Barbara, CA 93106.

Table 1

Examples of Problems Used in the Study

Standard Expressions (Problems 1-10)	Answer for Number Strategy	Answer for Equals Strategy	Answer for Operator Strategy
2 + 3	5	3	3
2 + 3 +	5	3	5
2 + 3 + 7	12	7	7
Non-Standard Expressions (Problems 11-23)	**Answer for Ignore Strategy**	**Answer for Reset Strategy**	**Answer for Increment Strategy**
2 + =	2	0	4
2 + + =	2	0	4 or 6 or 8
2 x =	2	0	4
2 x x =	2	0	4 or 8 or 16
2 + x 3 =	5 or 6	3 or 0	12 or 15
Chain Expressions (Problems 24-25)	**Answer for Left Strategy**	**Answer for Multiply Strategy**	**Answer for Other Strategy**
2 + 3 x 7	35	23	(varies)
2 x 3 + 7	13	13	(varies)

Table 2

Number of Subjects From Each Treatment Group Who Were Classified as Using a Number, Equals, or Operation Strategy for When to Evaluate an Expression -- Problems 1 to 10

	Evaluation Strategy		
Group	Number	Equals	Operator
Line Group (n=24)	2	6	16
Stack Group (n=24)	2	10	12
Control (n=24)	8	9	7

Note. - Chi-square test indicates significant difference among strategies used by the three groups, $p < .02$.

Table 3

Number of Subjects From Each Treatment Group Who Were Classified as Using an Ignore, Reset, or Increment Strategy for How to Evaluate a Non-Standard Expression -- Problems 11 to 23

	Evaluation Strategy		
Group	Ignore	Reset	Increment
Line Group (n=24)	10	6	8
Stack Group (n=24)	15	3	6
Control (n=24)	18	0	6

Note. - Chi-square test indicates significant differences among strategies used by the three groups, $p < .05$.

Table 4

Number of Subjects From Each Treatment Group Who Were Classified as Using a Left-to-Right, Multiply-before-Add, or Other Strategy for How to Evaluate a Chain of Expressions -- Problems 24 to 25

	Evaluation Strategy		
Group	Left-to-Right	Multiply-before-Add	Other
Line Group (n=24)	18	2	4
Stack Group (n=24)	17	1	6
Control (n=24)	17	3	4

Note. - No significant differences among groups in strategy usage.

Figure Captions

Figure 1. The Four Function Calculator

Figure 2. The Line Model

Figure 3. The Stack Model

Figure 4. Examples of Problems for Line, Stack and No Model Groups

Figure 1. The Four Function Calculator

In this experiment, you should pretend that you have just been given a simple handheld calculator. Your calculator looks like the one shown below. Notice that it has 10 digit keys (labeled 0 through 9); 4 function keys for addition (+), subtraction (-), multiplication (x), and division (÷); 1 equals key (=); and a digital display which presents numbers. You can press keys in a certain order, and the answer will appear in the display.

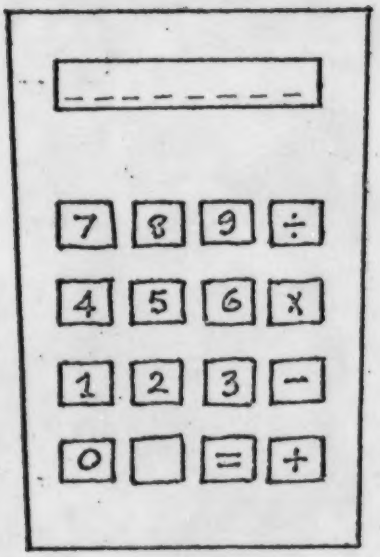


Figure 2. The Line Model

As you can see, the diagram on the previous page shows what the calculator looks like on the outside. We also want you to think about what goes on inside the calculator as you press each key. There are two internal parts of the calculator that we ~~what~~ want you to consider, as shown below. These are the display-register (which is also seen on the outside), and the main-register that contains lines and boxes for the numbers and the operator symbols to be registered respectively. Each line may hold one number up to 8 digits in length. Each box may hold one operator symbol (such as +, -, x, or ÷). It is possible to copy a number from the display-register to one of the slots in the main-register. You must fill in the leftmost slot or box first and work from left to right within the main-register. Only the display-register is visible during calculations and thus this is where the answer will be shown.

Figure 3. The Stack Model

As you can see, the diagram on the previous page shows what the calculator looks like on the outside. We also want you to think about what goes on inside the calculator as you press each key. There are four internal parts of the calculator that we want you to consider, as shown below. The four parts are the display-register (which is also seen on the outside), the X-register, the Y-register, and the operator-register.

The operator-register can hold one operator symbol (such as +, -, x, or ÷) at a time. If you press a function key, the previous operator symbol sitting in this register gets erased and the new one you just pressed goes in the register.

The three other registers, the display-register, the X-register, and the Y-register can hold numbers up to 8 digits in length. These registers can carry one number at a time. If there is already a number in the register and if you put a new number in the same register, the old one sitting there will be erased. However, you may copy a number from one register to another (such as from the display-register to the X-register or from the X-register to the Y-register). Among the registers only the display-register is visible during calculations and thus this is where the answer will be shown.

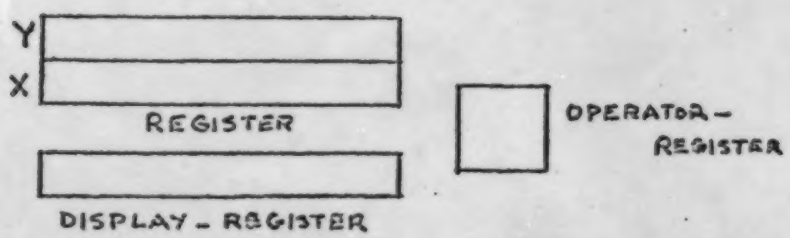


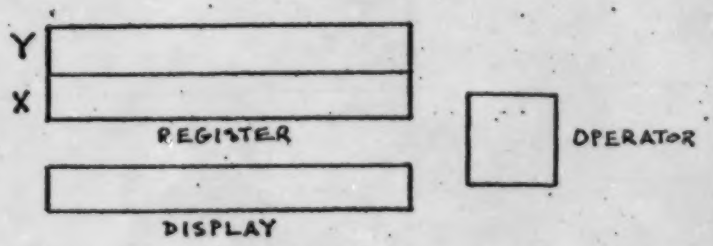
Figure 4. Examples of Problems for Line, Stack, and No Model Groups

Line Group

2 + + =

Stack Group

2 + + =



No Model Group

2 + + =

CHAPTER 6

EFFECTS OF INSTRUCTION ON REMEDIATION OF BUGS AND PROBLEM SOLVING FOR THE BASIC LANGUAGE

Note

This research will be submitted for publication as follows:

Bayman, P., & Mayer, R. E. Effects of instructional method on users' mental models for BASIC programming statements. Journal of Educational Psychology, in preparation.

The main purpose of the present study is to determine whether the format of instruction for a computer programming language can influence the learner's mental model of the computer. The user's mental model refers to the user's conception of what happens inside the computer between input and output.

Previous research (Bayman & Mayer, in press) found that when college students study BASIC through a standard manual coupled with hands-on experience, they develop several fundamental misconceptions regarding the meaning of programming statements. For example, about one half of the users thought that LET A = B + 1 resulted in the computer storing an equation in memory. In addition, about one-third thought that PRINT C meant that the computer would print the letter C on the screen, and about 10% thought that READ A meant that the computer would print the value of A on the screen. It should be pointed out that subjects were able to solve the problems in the manual, and were able to successfully engage in hands-on contact with the computer. However, in spite of mastering the problems in the manual, most users developed mental models that were either incorrect or incomplete.

Developing an appropriate mental model in conjunction with one's first programming language may be critical to one's success in transfer and in using the language creatively. Hence, the present study is aimed at testing whether some instructional methods can help learners to develop useful mental models--i.e., the present study examines whether mental models can be taught as part of normal instruction.

Several researchers have recommended that novices should learn their first language in a way that allows them to spontaneously build a mental model of the system (Carroll & Thomas, 1982; Moran, 1981; Young, 1981). Mayer (1981) has also summarized research concerning the role of explicit mental models in learning computer programming.

A task analysis of the BASIC language (Mayer, 1979) reveals that each statement can be described as a list of "transactions"--i.e., an action performed on some object in some computer location. For example, the following transactions are involved in the statement LET A = 0:

1. Find the number in memory space A.
2. Erase the number in memory space A.
3. Find the number to the right of the equals sign in the statement.
4. Write this number in memory space A.
5. Find the next statement in the program.

Hence, through a set of transactions, we can clearly specify the actions that occur within the computer as any statement is executed. The present study attempts to explicitly teach the transactions involved in each statement, in order to determine whether such explicit instruction leads to less misconceptions and better criterion performance.

Method

Subjects. The subjects were 95 college students at the University of California, Santa Barbara. The subjects had no previous experience or knowledge about computer programming.

Design. The subjects were randomly distributed into five groups: (1) Standard Group, in which the subjects studied BASIC from a standard manual; (2) Summary Transaction Group, in which subjects studied from a manual that included a verbal summary of the key actions carried out for each BASIC statement; (3) Transaction Group, in which subjects studied from a manual that included a verbal description of each action carried out by the computer for each BASIC statement; (4) Diagram Group, in which subjects studied from a manual that included diagrams showing each of the actions carried out by the

computer for each BASIC statement; (5) Transaction/Diagram Group, in which subjects studied a manual that contained both a verbal description and a diagram description for each BASIC statement.

There were 19 subjects in each group; however, data from one subject in Group 5 was eliminated because that subject failed to follow directions.

Materials. The materials consisted of a subject questionnaire, five instructional manuals, two mastery tests, and four posttests.

The subject questionnaire was typed onto an 8½ x 11 inch sheet of paper, and consisted of questions concerning: the subject's demographic characteristics, such as age, sex, and academic major; the subject's mathematics background, such as geometry, algebra and calculus courses the individual had taken in high school or college; the subject's ability, such as SAT-Verbal score, SAT-Quantitative score, and grade point average; the subject's programming background, such as whether the individual had ever operated a programmable calculator, typed at a computer terminal, etc.

There were five kinds of manuals: standard manual, summary transaction manual, transaction manual, diagram manual, and transaction/diagram manual. Each manual was typed double-spaced onto 8½ x 11 inch sheets of paper.

The standard manual was adopted from a self-instructional, mastery manual called BASIC in Six Hours (Marcus, 1980) which is widely used in teaching BASIC to students in the Microcomputer Laboratory of the University of California, Santa Barbara. The manual consisted of two lessons. The first lesson covered statement execution in the immediate mode as well as some introductory information for preparing a program. The statements and commands introduced in the first lesson were: PRINT, LET, IF-THEN statements and NEW, LIST and RUN commands. The second lesson covered programming and included some sample programs. The statements introduced in the second lesson were

GOTO, IF-GOTO, INPUT, READ and DATA. The standard manual provided information concerning the syntax for each statement; examples provided information concerning what the output would be for a given input. For example, PRINT X results in a number appearing on the screen.

The summary transaction manual was identical to the standard manual except for some added information. For each statement, a description of the key actions that occur within the computer was given. For example, LET X = A + B means that computer "stores in space X the sum of the number in A and the number in B."

The transaction manual was identical to the standard manual except for some added information. For each statement, a detailed description of each action that occurs within the computer was given. For example, LET X = A + B, means: "find the number in space A but do not erase it," "find the number in memory space B but do not erase it," "add those numbers together," "erase the old number in space X," "put the sum in space X," "go on to the next statement."

The diagram manual was identical to the standard manual except that some information was added. For each statement, a diagram of the inside of the computer was shown before and after the statement was executed. The diagram showed the status of an input stack, an output screen, a wait-run light, a program list, and a memory scoreboard. For example, for LET X = A + B, in the before diagram the number 5 was in space A, 4 in B and 3 in X; the after diagram showed 5 in A, 4 in B, and 9 in X.

The transaction/diagram manual was the same as the transaction manual and diagram manuals combined. For each statement both a verbal description of action and before/after diagrams were given.

There were two sets of mastery tests--one for lesson 1 and one for lesson 2. Each test was typed on a 8½ x 11 inch sheet of paper. Each test consisted of 5 questions based on the contents of the lesson. Each test had a 10-minute time limit. Several versions of each mastery test were available for each lesson.

There were four posttests: general criterion, fact, verbal specification, diagram specification. The general criteria that consisted of 18 problems, each typed on a 5 x 8 inch sheet, including six of each of the following types: interpretation problems presented a statement or program and asked the subject to write what was accomplished in English; generation problems presented a problem in English and asked the subject to write a statement or program to solve the problem; debugging problems presented a flawed statement or program and asked the subject to tell what was wrong.

The fact retention test consisted of twenty fill-in-the-blanks questions about information that was explicitly stated in the manuals. The test was typed on an 8½ x 11 inch sheet of paper.

The verbal specification task test consisted of ten 8½ x 11 inch sheets of paper, with a BASIC statement typed on each. The subject's job was to write, in plain English, the actions that the computer would carry out to execute each statement. Each sheet contained several blank lines, and subjects were instructed to write one action per line. The following statements were given: 70 LET a = B + 1; 60 DATA 5, 10, 13; 10 PRINT C; 30 LET D = 0; 80 GOTO 10; 20 READ B; 90 PRINT 'C"; 40 IF A > B THEN GOTO 99; 100 INPUT A; 50 IF A=B THEN PINT "THEY ARE EQUAL".

The diagram specification test consisted of ten 8½ x 11 inch pages, with a BASIC statement typed on each. In addition, the top of the page presented in diagram of the computer indication the contents of the input stack, the

contents of the memory scoreboard, the status of the run-wait light, the information printed on the output screen, and the pointer arrow's location in the program list. This diagram represented the state of the computer before the statement was executed. At the bottom of the page, there was a blank diagram for the subject to fill in in order to indicated the status of the computer after the statement is executed. The same 10 statements were used as for the verbal specification test.

Procedure. The experiment consisted of five one-hour sessions over a one to two week period. In the first session, subjects filled out the subject questionnaire, received a brief description of the study, and signed up for times for the following four sessions. In the second and third session, subjects worked on their respective manuals, following random assignment for treatment groups. Subjects studied the manuals at their own pace, one lesson at a session. Subjects were given the 10-minute mastery test after each lesson. If a subject missed more than one item, the errors were explained by the experimenter and the subject was asked to review the manual. Then, another form of the mastery test was given, and so on. During the final two sessions, the four posttests were given in the following order: general criterion, fact retention, verbal specification, diagram specification. Subjects answered the questions on the tests at their own pace.

Results

This section provides a preliminary analysis of the results with special focus on whether the instructional treatments (i.e., groups 2 through 5) increased problem solving performances (criterion test) and decreased misconceptions (verbal and diagram specification tests).

Do the groups differ in study time? Since the experimental manuals (Groups 2 through 5) contain more material than the standard manual, it is

likely that subjects in the experimental groups will require more study time. The mean times required to read the two lessons for Groups 1 through 5 were 43, 48, 49, 59 and 74 minutes, respectively. An analysis of variance revealed significant differences among the groups, $F(4,89) = 15.24$, $p < .001$.

Does explicit model instruction affect problem solving? Learning a mental model should enhance performance on the criterion test. The average correct on the criterion test was computed for each of the five treatment groups. An analysis of variance revealed no significant differences among the groups in criterion score, $F < 1$. Individual t-tests revealed no significant differences among the four experimental groups (i.e., Groups 2, 3, 4 and 5); therefore, these four groups were combined for subsequent analysis. The subjects in each group were divided into two ability levels--those scoring above 530 in SAT-Quantative were called "high ability" and those scoring at or below 530 were called "low ability." An analysis of variance was conducted with treatment (group 1 versus the combination of all experimental groups) and ability (high versus low) as between subject factors. The ANOVA revealed a significant aptitude treatment interaction (ATI) in which low ability students learned better when given explicit training in mental models (experimental groups) but high ability learners did better when given only the standard manual (group 1), $F(1,90) = 3.84$, $p < .05$. Apparently, high ability learners are able to generate their own useful models based on their past experience in mathematics tasks while low ability learners benefit from the instructor explicitly providing a model. These results are summarized in Table 1.

Insert Table 1 about here

Do the groups differ in retention of specific facts? Learning a mental model should not affect the learner's retention of specific facts, since the model is independent of the specific facts. Table 2 lists the percentage correct retention for the five groups. As can be seen, the experimental treatment does not enhance fact retention. An analysis of variance revealed no significant differences among the means, $F(4,89) = 1.74$.

Insert Table 2 about here

Does explicit training in mental models reduce the number of misconceptions? The experimental treatments (groups 2 through 5) should result in fewer misconceptions than the standard treatment. In order to test this idea, the verbal and diagram specification tests were scored. If the user generated the key actions, and no incorrect actions, for a statement, that statement was counted as correct. Table 3 shows the mean number of correct statements for subjects in each of the five treatment groups. As can be seen, the experimental groups tend to perform better than the standard group. An analysis of variance comparing the standard group against the four experimental groups revealed a significant difference in overall number of correct statements, $F(1, 89) = 5.84$, $p < .05$.

Insert Table 3 about here

Discussion

This study provides promising indications that mental models can be explicitly taught to novices. In particular, explicit teaching of mental models seems most effective for low ability learners. This result is

consistent with earlier findings by Mayer (1975). In addition, the transaction treatment (group 3) seems to be the most efficient instructional procedure because it requires very little additional time while producing the largest gains in problem solving and verbal specification performance. This study compliments earlier research (Bayman & Mayer, in press) showing that novices tended to acquire many misconceptions. Apparently, fairly modest alterations in the instructional program (e.g., adding a discussion of transactions) can greatly reduce user misconceptions. The reduction of misconceptions is especially important for users who plan to continue learning about computer programming, since they will need to build on their existing conceptions. Further work is needed to determine the long-term effects of user misconceptions, and the effects of remediation.

References

Bayman, P. & Mayer, R. Diagnosis of beginning programmer's misconceptions of BASIC programming statements. Communications of the ACM, in press.

Caroll, J.M. & Thomas, J.C. Metaphor and the cognitive representation of computing systems. EEE Transactions on Systems, Man, and Cybernetics, 1982, 12, 107-116.

Marcus, J. Basic in Six Hours: A Self-Instruction Text. Santa Barbara: Microcomputer Laboratory, 1980.

Mayer, R.E.. Different problem-solving competencies established in learning computer programming with and without meaningful models. Journal of Educational Psychology, 1975, 67, 725-734.

Mayer, R.E. A psychology of learning BASIC. Communications of the ACM, 1979, 11, 589-593.

Mayer, R.E. The psychology of how novices learn computer programming. Computing Surveys, 1981, 13, 121-141.

Moran, T.P. An applied psychology of the user. Computing Surveys, 1981, 13, 1-11.

Young, R.M. The machine inside the machine: User's models of pocket calculators. International Journal of Man-Machine Studies, 1981, 15, 51-85.

Table 1

Percent Correct on the General Criterion Test for Experimental and Standard Subjects

Treatment Groups	Ability Level	
	Low	High
Experimental Groups	58%	65%
Standard Group	48%	74%

Table 2

Percentage Correct on the Fact Retention Test for Five Groups

Treatment Group	Percent Correct
Standard	72%
Summary Transaction	74%
Transaction	70%
Diagram	62%
Transaction/Diagram	72%

Table 3

Percent of statements Described Correctly by Subjects in Five Treatment Groups

Treatment Group	Percent Correct
Standard	58%
Summary Transaction	64%
Transaction	71%
Diagram	66%
Transaction/Diagram	66%

APPENDIX 1

LIST OF PUBLICATIONS AND PAPERS

Below is a list of published products of this project, including works that are under review or in preparation.

Mayer, R. E. The psychology of how novices learn computer programming. Computing Surveys, 1981, 13, 121-141. (Also reprinted in: Curtis, B. Human Factors in Software Design. Los Angeles: IEEE Computer Society Press, 1981. Also reprinted in the Japanese magazine, Byte, 1983.)

Mayer, R. E. Contributions of cognitive science and related research on learning to the design of computer literacy curricula. In R. J. Seidel, R. E. Anderson & B. Hunter (Eds.), Computer Literacy: Issues and Directions for 1985. New York: Academic Press, 1982, 129-161.

Mayer, R. E., & Bayman, P. A psychology of calculator languages: Describing computer concepts to novices. Communications of the Association for Computing Machinery, 1981, 24, 511-520.

Mayer, R. E. Users' misconceptions of BASIC computer programming statements. Communications of the Association for Computing Machinery, in press.

Bayman, P., & Mayer, R. E. Instructional manipulation of users' mental models for calculators. International Journal of Man Machine Studies, under review.

Mayer, R. E., & Bayman, P. Users misconceptions concerning the operation of electron calculators. Cognition & Instruction, under review.

Bayman, P., & Mayer R. E. Users' misconceptions of BASIC programming statements following hands-on experience. Cognition & Instruction, under review.

Bayman, P. Effects of instruction on beginning programmers' mental models for BASIC. Manuscript in preparation.

Below is a list of paper presentations concerning this project:

Mayer, R. E., & Bayman, P. Analysis of student's intuitions about the operation of electronic calculators. American Educational Research Association, Los Angeles, April 13-17, 1981.

Mayer, R. E. Making the most of micro-computers in a psychology department. American Psychological Association, Los Angeles, August 24-28, 1981.

Mayer, R. E. Software Psychology. University of San Francisco, Special Lecture Series in Computer Science, September 25, 1980.

Mayer, R. E. Psychology of Computer Programming for Novices. NSF Invitational Conference on Computer Literacy. Reston, VA, December 19, 1980.

Mayer, R. E. Teaching Computer Languages. Claremont Graduate School, Invitational Conference on Applied Cognitive Psychology, Claremont, CA, April 2, 1982.

Mayer, R. E., & Bayman, P. Users' mental models for BASIC programming statements. Psychonomic Society, Minneapolis, November 11-13, 1982.